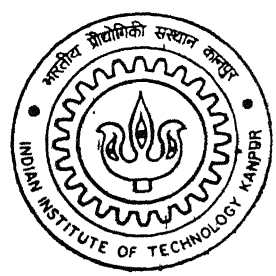
ALGORITHMS FOR NEURO-FUZZY SYSTEMS

By

G V Bhavani Sankar



DEPARTMENT OF ELECTRICAL ENGINEERING

Indian Institute of Technology Kanpur

JANUARY, 2002

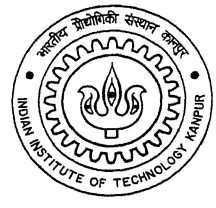
ALGORITHMS FOR NEURO-FUZZY SYSTEMS

A Thesis submitted
in partial fulfilment of the requirement
for the degree of

MASTER OF TECHNOLOGY

by

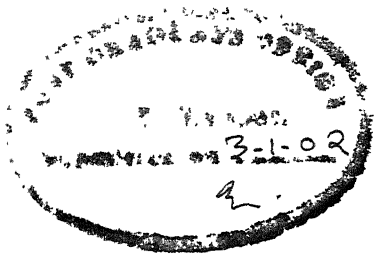
G V Bhavani Sankar



To the

Department of Electrical Engineering
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

January 2002



CERTIFICATE

This is to certify that the work contained in this thesis entitled **"Algorithms for Neuro-Fuzzy Systems"**, by G V Bhavani Sankar, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr P K Kalra
Professor
Department of Electrical Engineering
Indian Institute of Technology, Kanpur

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Dr. P K Kalra, my thesis supervisor, for his inspiring guidance and sincere supervision, not only during the thesis work, but during the complete academic program both inside and outside the lab I am thankful to him for the complete freedom afforded to me in my work as well as for the excellent facilities in the lab. The completely open atmosphere for exchanging views in the lab, not only with him but also with others, was what helped me in achieving what I could, in this short span of time

I would also like to express my thanks to Prashant and Madhav, who were always there to help, discuss and share their experience in this field. I am thankful to Miss. Manu for her help.

I am grateful to Manoj, Saleem, Balaji, Ashesh, Maj. Jayesh, Maj Vaid, Maj. Mohan Kumar and Maj. Patil for their words of encouragement, co-operation during my period of stay in IIT Kanpur Last but not the least; I would like to thank my parents for their constant support and encouragement.

G V Bhavani Sankar

ABSTRACT

An attempt has been made to implement neural networks comprising of OR/AND neurons as the basic functional components. It has been observed that instead of using a single large network for solving a problem, the use of several smaller networks on properly divided data improves the learning performance, in general. Neuro-fuzzy systems (or rule-based systems) have been considered, as single architectures, to incorporate proper division of data and the smaller networks that approximate the function in each of the smaller divisions. The two different types of neuro-fuzzy systems considered are ANFIS and CNFS. The salient features of these neuro-fuzzy systems are heuristic initialization of parameters and parameter estimation using hybrid and compensatory learning algorithms.

The problem of structure identification, which concerns partitioning the input space and determining the number of fuzzy rules for a specific performance, has been studied. An off-line rule-based clustering algorithm is proposed to cluster the input data and to determine the number of rules. A dynamic approach for generation of rules has also been considered, where an on-line self-constructing algorithm is used. The algorithms developed are validated on several benchmarking problems of function approximation and classification.

INDEX

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Several attempts have been reported to understand and model the capabilities of human brain. Some of these are Expert Systems, Artificial Neural Networks, Fuzzy Logic, Genetic algorithms etc. These algorithms represent different levels of human information processing. However, each model may not perform all functions performed by brain independently. Every intelligent technique has particular computational properties (e.g. ability to learn, explanation of decisions) that make them suited for particular problems and not for others. For example, while neural networks are good at recognizing patterns, they are not good at explaining how they reach their decisions. Fuzzy logic systems, which can reason with imprecise information, are good at explaining their decisions but they cannot automatically acquire the rules they use to make those decisions. These limitations have been a central driving force behind the creation of intelligent hybrid systems where two or more techniques are combined in a manner that overcomes the limitations of individual techniques. In the present work the computational power of Neuro-Fuzzy Systems, which are Fuzzy Rule-Based systems implemented in the framework of neural networks, has been explored.

1.1 OR/AND Neural Network as an alternative to ANN

The classic neural network plays the role of a universal approximator that gives rise to a purely descriptive model. An attempt has been made to implement the OR/AND neural network in which because of the normative aspects of the logical connectives, the weight

spaces will be unit hyper cubes, reducing the search space, as an alternative to Artificial Neural Networks to overcome some of the problems of ANN like slow convergence, local minima etc.

The OR/AND Network is a neural network model based on logical connectives. The basic processing unit consists of two types of generic OR and AND neurons structured into a three layered topology.

The pure AND or OR operations, if used in aggregation, cannot cope well with experimental data [1]. In this thesis, an attempt has been made to alleviate this problem using several new compensatory logic operators, whose characteristics constitute a mixture of AND and OR features of the pure connectives. A weight factor [0,1] has used to express the grade of compensation. Different functions for the AND and OR operations have been tried. Several architectures of neural networks comprising these neurons as their basic functional components were implemented. The inputs to the neurons are weighted through connections to differentiate between particular levels of impact that the individual inputs might have on the result of the aggregation. An attempt has been made to incorporate the global approximation capabilities, via a multilayer architecture of the logic networks.

Learning in the above OR/AND neuron is addressed as a problem of supervised training which concerns a series of modifications of the connections (parametric learning) using gradient descent algorithm. The specificity of the logical connectives is captured by the OR/AND neuron within its supervised learning.

The network architectures comprising these neurons are experimented with an additionally furnished nonlinear element such as the sigmoid function placed in series with the OR neuron situated at the output layer, while approximating non-linear functions. Since

the AND and OR operations themselves are nonlinear functions of the inputs, it was expected that they alone can realize any nonlinear function, with lesser architectures as compared to Artificial Neural Networks

1.2 Objective

The complexity of a network depends on the number of layers and the number of neurons. Because of the requirement of high complexity, this neural network model may require a large number of neurons, when used to solve highly nonlinear problems. The complexity of this network can be reduced by improving the learning algorithms or by virtually reducing the complexity of the problem considered.

When the input data is partitioned into smaller regions, the result is a reduced local complexity of the problem, in each of the regions. Then smaller neural networks may be used for each of the partitions to solve the problem locally. Since each network is small and they can be trained simultaneously, the total learning time can be reduced significantly. The limiting case is obtained when in each partition only a linear element will be able to approximate the given function. The objective of this thesis is to solve the problem of incorporating the partitioning algorithm and the smaller neural networks in a single architecture.

Neuro-Fuzzy Systems (or Rule-Based Systems) are considered to solve this problem. They can be called inherent partition systems because they partition the input data and use if-then rules, whose premise parts decide which input belongs to which partition space and a simple or complex neuron in the consequent part approximate the function within the partition. Fuzzy logic reasoning also makes the output a continuous function. An

additional advantage of these systems is their ability to represent a given system in the form of if-then rules.

When the consequent part consists of only linear terms, the network architecture is referred to as ANFIS, standing for Adaptive Network-Based-Fuzzy Inference System. In the case of ANFIS each input space is partitioned into a prespecified number of fuzzy regions, denoted by membership function labels, like "small", "medium" and "large" and so on, depending on the number of partitions selected. A bell shaped membership function has also been used as the consequent. A compensatory reasoning mechanism has been used with the neuro-fuzzy system, making it the Compensatory Neuro-Fuzzy System.

The problem associated with both these algorithms is the selection of number of rules. This problem has been attacked by two approaches. One way is to cluster the input space and considering as many rules as the number clusters. A rule-based clustering algorithm has been proposed, to cluster the input data. In the other approach, rules are generated dynamically as and when their necessity is felt.

1.3 Organization of thesis

Chapter 2 describes the AND/OR neural network in detail and outlines its shortcomings. Chapter 3 introduces the two rule-based systems considered in this thesis ANFIS and CNFS, along with their learning formulae. Chapter 4 describes the newly proposed rule-based clustering technique and how it can be integrated with the neuro-fuzzy systems, to improve their performance. The dynamic incorporation of rules also was discussed in this chapter. Chapter 5 presents the results for the different problems considered on the different algorithms. Chapter 6 concludes the present work and outlines the aspects of further development.

CHAPTER 2

OR/AND NEURON BASED NEURAL NETWORKS

2.1 Introduction

The OR/AND neuron proposed by Hirota *et al.* [1] takes the form of a logic operator whose characteristics constitute a mixture of AND and OR features of the pure connectives. The contribution of these two basic components can then be conveniently modeled through an auxiliary parameter of the model. This model takes on the form

$$y = \left[AND(x_i, x_j)\right]^{1-\gamma} \left[OR(x_i, x_j)\right]^{\gamma} \qquad \ldots(2.1)$$

where x_i and x_j stand for the weighted inputs fed to the neurons, y denotes the result of aggregation, while γ is a weight factor used to express the grade of compensation, $\gamma \in [0,1]$. The value equal 0 yields no compensation while set to 1/2 it produces the highest compensation between the AND and OR forms of the overall aggregation.

Another version of the connective exhibiting a similar compensatory nature can be realized in the form of the following convex combination,

$$y = AND(x_i, x_j) + (1-\gamma)OR(x_i, x_j) \qquad \ldots(2.2)$$

The above two-argument operators can be directly expanded into many-variable versions.

2.2 The AND and OR Neurons

The logic-based neurons can be regarded as multivariable nonlinear transformations between unit hyper cubes, $[0,1]^n \rightarrow [0,1]$. The AND neuron aggregates input signals

$x = [x_1, x_2, ...x_n]$ by first combining them individually with the connections (weights) $w = [w_1, w_2, ..w_n] \in [0,1]^n$ and afterwards globally ANDing these results,

$$y = AND(x; w) \qquad \ldots(2.3)$$

The structure of the OR neuron is dual to that reported for the AND neuron, namely,

$$y = OR(x; w) \qquad \ldots(2.4)$$

The AND and OR neurons realize "pure" logic operations on the inputs. The role of the connections is to differentiate between particular levels of impact that the individual inputs might have on the result of the aggregation Higher values of connections in the OR neuron emphasize a stronger influence that the corresponding inputs pose on the output of the neuron. The opposite weighting effect takes place in case of the AND neuron, the values of w_i close to one make that influence of x_i almost negligible The specific numerical form of this relationship depends upon the triangular norms being utilized. For the AND neuron the lower limit can be interpreted as the initial confidence of simultaneous satisfaction of inputs. The upper bound on the OR neuron comes as a consequence of the belief that the complete satisfaction of all inputs should lead to the highest level of this aggregation y that is however less than 1.

The different types of t-norms and t-co norms [2] used are presented below. The formulae given below are for two inputs; they can be easily extended for multi inputs.

T-Norms:

1. Product, ab
2. Derived from averaged sum OR, $\frac{1}{2}\left(1-\left(\frac{1}{2}(1-a)+\frac{1}{2}(1-b)\right)\right)$
3. Derived from probabilistic sum OR, $1-(1-a)-(1-b)+(1-a)(1-b)$
4. Dombi, $\left(1+\left(\left(\frac{1}{a}-1\right)^{p}+\left(\frac{1}{b}-1\right)^{p}\right)^{\frac{1}{p}}\right)^{-1}$
5. Frank, $\log_{p}\left[1+\frac{(p^{a}-1)(p^{b}-1)}{p-1}\right]$
6. Hamacher, $\frac{ab}{p+(1-p)(a+b-ab)}$
7. Schweizer & Sklar2, $1-\left[(1-a)^{p}+(1-b)^{p}-(1-a)^{p}(1-b)^{p}\right]^{\frac{1}{p}}$
8. Schweizer & Sklar3, $\exp\left(-\left(|\ln a|^{p}+|\ln b|^{p}\right)^{\frac{1}{p}}\right)$
9. Schweizer & Sklar4, $\frac{ab}{\left[a^{p}+b^{p}-a^{p}b^{p}\right]^{\frac{1}{p}}}$

T-Co Norms:

1. Derived form product AND, $(1-a)(1-b)$
2. Averaged Sum, $\frac{1}{2}(a+b)$
3. Probabilistic Sum, $a+b-ab$

4. Dombi, $\left(1+\left(\left(\frac{1}{a}-1\right)^{p}+\left(\frac{1}{b}-1\right)^{p}\right)^{-\frac{1}{p}}\right)^{-1}$

5. Frank, $1-\log_{p}\left[1+\frac{\left(p^{1-a}-1\right)\left(p^{1-b}-1\right)}{p-1}\right]$

6. Hamacher, $\frac{a+b+(r-2)ab}{r+(r-1)ab}$

7. Schweizer & Sklar2, $\left[a^{p}+b^{p}-a^{p}b^{p}\right]^{\frac{1}{p}}$

8. Schweizer & Sklar3, $1-\exp\left(-\left(\left|\ln(1-a)\right|^{p}+\left|\ln(1-b)\right|^{p}\right)^{\frac{1}{p}}\right)$

9. Schweizer & Sklar4, $1-\frac{(1-a)(1-b)}{\left[(1-a)^{p}+(1-b)^{p}-(1-a)^{p}(1-b)^{p}\right]^{\frac{1}{p}}}$

The weighted mean was also considered as a compensatory logic operator,

$$y=\left(\sum_{i=1}^{n}w_{i}x_{i}^{p}\right)^{\frac{1}{p}} \qquad \ldots(2.5)$$

Here the modifiable degree of compensation is accomplished by changing the value of the power coefficient "p". For $p=-\infty$ it yields the minimum operator while $p=+\infty$ it behaves as the maximum operator.

The derivatives used in updating the weights are presented in Appendix A.

2.3 Architecture of the OR/AND neuron

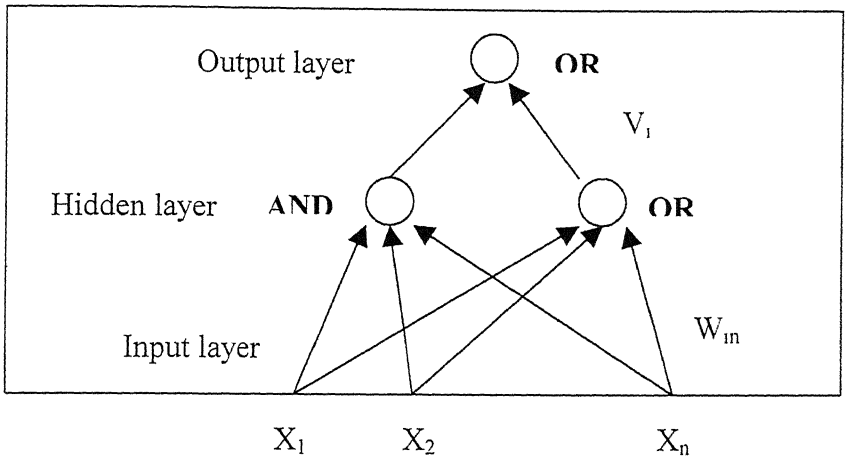


Fig 2.1 Architecture of OR/AND neuron

The implemented OR/AND neuron constitutes a three layer network and is constructed by arranging the above mentioned AND and OR neurons into a structure displayed in Fig. 2.1. Due to the strong functional cohesion existing between the neurons, it is called OR/AND neuron. The relevant detailed formulas describing this architecture read as,

$$y = OR([z_1, z_2]; V)$$
$$z_1 = AND(x; W_1) \text{ and } z_2 = OR(x; W_2) \qquad \ldots(2.6)$$

with $V = [v_1, v_2]$, $W_i = [w_{i1}, w_{i2}, \ldots w_{in}]$, $i = 1,2$, being the connection weights of the corresponding neurons.

The pure AND and OR characteristics produced at the level of the hidden layer are then combined by the OR neuron constituting the output layer. Changing the connection weights between hidden and output layer an entire range of intermediate logical characteristics of the structure spread between the AND-like and OR-like type of functional behavior can be obtained. The connections V supply a necessary flexibility required to

achieve various levels of compensation between the AND and OR character of the neuron. The OR neuron situated at the output layer was additionally furnished with a nonlinearity element such as a sigmoidal function placed in series with the previous architecture.

The functions defined by equations (2.1) and (2.2) have also been implemented in place of the OR neuron at the output layer.

2.4 Learning

Learning in the above OR/AND neuron was addressed as a problem of supervised training, which concerns a series of modifications of the connections (parametric learning) using gradient descent algorithm. The successive updates of the connections $W = \lfloor w_{ij} \rfloor$ and $V = [v_1, v_2]$ within this training are controlled by the gradient of the predetermined performance index, in our case the mean squared error performance criterion (Q). The formulae for weight updation considering the mean squared error performance criterion and admitting an on-line type of learning, (the relevant updates are performed after a presentation of each pair of the elements of the training data set.) are presented below. Considering the OR/AND neuron without the nonlinear element, we get,

$$\frac{\partial Q}{\partial w_{ij}} = -2(t\arg et - y)\frac{\partial y}{\partial z_i}\frac{\partial z_i}{\partial w_{ij}}$$

$$\frac{\partial Q}{\partial v_i} = -2(t\arg et - y)\frac{\partial y}{\partial v_i} \qquad \ldots(2.7)$$

$$i = 1,2,\ j = 1,2,\ldots n \text{ where } Q = (t\arg et - y)^2.$$

Further detailed computations of the above derivatives are presented for the specified triangular norms used to develop the neurons, in Appendix A.

The equation for the nonlinear sigmoid element placed in series with the OR neuron of the output layer can be written as,

$$y = \frac{1}{1 + \exp[-(outp - m)\sigma]} \qquad ..(2.8)$$

with *outp* being the output of the OR neuron, or one of the other compensation operations in the output layer. Both the parameters of the nonlinearity (m and σ) were subjected to changes during the learning of the neuron. The learning formulae become slightly modified as compared to above derived,

$$\begin{aligned}
\frac{\partial Q}{\partial w_{ij}} &= -2(t\arg et - outp)outp(1 - outp)\sigma \frac{\partial y}{\partial z_i}\frac{\partial z_i}{\partial w_{ij}} \\
\frac{\partial Q}{\partial v_i} &= -2(t\arg et - outp)outp(1 - outp)\sigma \frac{\partial y}{\partial v_i} \\
\frac{\partial Q}{\partial m} &= -2(t\arg et - outp)outp(1 - outp)(-\sigma) \\
\frac{\partial Q}{\partial \sigma} &= -2(t\arg et - outp)outp(1 - outp)(y - m)
\end{aligned} \qquad ...(2.9)$$

2.5 Network architectures for representing a global type of aggregation

The OR/AND neuron was also considered as a part of a more extended architecture that is aimed at modeling a global character of aggregation carried out by the logical connectives. If we denote the resulting output of a multilayered architecture,

$$y = F(x_1, x_1) \qquad ...(2.10)$$

Both the values (x_1, x_1) as well as the computed value y pertain to one training pattern. The OR/AND neuron provides with a nonlinear realization of F. In this setting, OR/AND neurons will be playing a role of basic functional blocks of the multi-layered architecture. These architectures are tested on nonlinear function approximation problems.

2.6 Disadvantages of the OR/AND Neural Network

Through the simulations it has been observed that although the number of neurons and hidden layers required are less than that required for standard Neural Network, the reduction was not significant. The standard neural networks requires a 2-2-1 architecture to solve the XOR problem, where as only one neuron of the OR/AND network can solve the problem. For the SinxSiny problem also, when the standard network architecture that solved was 2-25-25-1 [3], the OR/AND neural network solved it using architecture of 2-12-8-1, without the nonlinear element. Since the number of hidden layers was two, learning was very slow, taking 83,448 iterations.

The other major disadvantage is the initialization of the network weights. Since they are randomly generated, the convergence largely depends on the initialization. With some weight combinations, the network may get trapped in local minima and may take very long time to come out of it. So, ways are required to make use of the data to initialize the weights heuristically, so that convergence is faster and guaranteed.

To overcome these problems the SinxSiny data was divided into 16 smaller data sets and each in turn was presented to a smaller network of architecture 2-1. Each of these networks was able to approximate the portion of the data presented in 1499 epochs. This indicates the need for networks that can partition the data and then approximate the function in each of the partitions. If-then rules can be one of the approaches to associate each input space partition with a smaller network. This gives rise to the idea of algorithms based on rules, the discussion of the following chapter.

CHAPTER 3

NEURO-FUZZY SYSTEMS (RULE-BASED SYSTEMS)

3.1 Introduction

In the field of artificial intelligence, there are various ways to represent knowledge. The most common way is to represent human knowledge into natural language expressions of the type,

$$IF\ premise(antecedent)THEN\ conclusion(consequent).$$

The above form is commonly referred to as the IF-THEN rule-based form. It typically expresses an inference such that if we know a fact (premise, hypothesis, antecedent), then we can infer, or derive, another fact called a conclusion (consequent). When the premise and consequent parts of the if-then rules are expressed as fuzzy membership functions, the resulting rule-base is called fuzzy rule-base and along with the inference mechanism, the total system is called fuzzy inference system.

A fuzzy inference system employing fuzzy if-then rules can model the qualitative aspects of human knowledge and reasoning processes without employing precise quantitative analyses. Fuzzy inference systems are good at explaining their decisions but they cannot automatically acquire the rules they use to make those decisions. Neural networks are used to tune membership functions of fuzzy systems that are employed as decision-making systems for increasing adaptability. Although fuzzy logic can encode expert knowledge directly using rules with linguistic labels, it usually takes a lot of time to design and tune the membership functions, which quantitatively define these linguistic labels. Neural network learning techniques can automate this process and substantially

reduce development time and cost while improving performance. When neural network techniques are used to train the parameters of the fuzzy membership functions, the resulting hybrid systems are known as neuro-fuzzy systems.

There are two different neuro-fuzzy systems, and the differences come from the specification of the consequent part. Jang's Adaptive Neuro-Fuzzy Inference System (ANFIS) [4] considers a crisp or linear function in the consequent part; while in the case of Zhang's Compensatory Neuro-Fuzzy System (CNFS) [5]; the consequent part is a monotonically non-decreasing or bell-shaped membership function.

3.2 Fuzzy If-Then Rules and Fuzzy Inference Systems

Fuzzy if-then rules or fuzzy conditional statements are expressions of the form IF A THEN B, where A and B are labels of fuzzy sets characterized by appropriate membership functions. Due to their concise form, they are often employed to capture the imprecise modes of reasoning that play an essential role in the human ability to make decisions in an environment of uncertainty and imprecision. Consider an example,

IF x is A THEN y is B

where A and B are linguistic variables, A and B are linguistic values or labels that are characterized by membership functions.

Fuzzy inference systems are also known as fuzzy-rule-based systems, fuzzy models or fuzzy associative memories. A fuzzy inference is composed of five functional blocks:

- A fuzzification interface which transforms the crisp inputs into degrees of match with linguistic values;
- A rule base containing a number of fuzzy if then rules;
- A data base which defines the membership functions of the fuzzy sets used in the fuzzy rules;
- A decision-making unit which performs the inference operations on the rules;
- A defuzzification interface which transforms the fuzzy results of the inference into a crisp output.

The rule base and the database are jointly referred to as the knowledge base. The steps of fuzzy reasoning (inference operations upon fuzzy if-then rules) performed by fuzzy inference system are:

- Compare the input variables with the membership functions on the premise part to obtain the membership values (or compatibility measures) of each linguistic label. (This step is called Fuzzification).
- Combine (through a specific T-norm operator, usually multiplication or min.) the membership values on the premise part to get the firing strength (weight) of each rule.
- Generate the qualified consequent (either fuzzy or crisp) of each rule depending on the firing strength.
- Aggregate the qualified consequents to produce a crisp output. (This step is called Defuzzification.

Depending on the type of fuzzy reasoning and fuzzy if-then rules employed, fuzzy inference systems can be classified into two types, namely

1. The output of each rule is a linear combination of input variables plus a constant term, and the final output is the weighted average of each rule's output.
2. Outputs are fuzzy membership functions. The overall output is calculated by applying any of the defuzzification methods

3.3 Adaptive Network-Based-Fuzzy Inference System (ANFIS)

When the outputs are linear combinations of the input variables, the resulting Neuro-Fuzzy System is known as ANFIS.

3.3.1 Adaptive Networks

An adaptive network, as its name implies, is a network structure consisting of nodes and directional links through which the nodes are connected. Moreover, part or all of the nodes are adaptive, which means their outputs depend on the parameters pertaining to these nodes, and the learning rule specifies how these parameters should be changed to minimize a prescribed error measure.

In an adaptive network, each node performs a particular function on incoming signals as well as a set of parameters pertaining to this node. The node can be either a square or a circle node. A square node has parameters while a circle node has none. The parameter set of an adaptive network is the union of the parameter sets of each adaptive node. In order to achieve a desired input-output mapping, these parameters are updated according to given training data and a learning procedure described later.

.3.2 ANFIS Architecture

In ANFIS, the input space is partitioned according to the number of membership unctions selected for each of the inputs. If the same number of membership functions is elected for all the inputs, the number of rules (*nr*) is given by "the number of membership functions for each input (*nmf*) raised to the power of the number of inputs (*ni*)".

$$\text{i.e , } nr = (nmf)^{ni} \qquad \ldots(3.1)$$

After the number of membership functions for each of the inputs is known, the input space is partitioned accordingly, and one rule in each of those partitions will be used to approximate the function in that region. In order to modify the parameters associated with the rules, adaptive network concepts are utilized.

Consider a fuzzy inference system with two inputs x and y and one output z, assuming the rule-base contains two type-3 fuzzy rules,

Rule 1: *If* x is A_1 and y is B_1, then $f_1 = p_1x + q_1y + r_1$ ...(3.2)

Rule 2: *If* x is A_2 and y is B_2, then $f_2 = p_2x + q_2y + r_2$ ...(3.3)

The corresponding ANFIS architecture is shown in Fig. 3.1

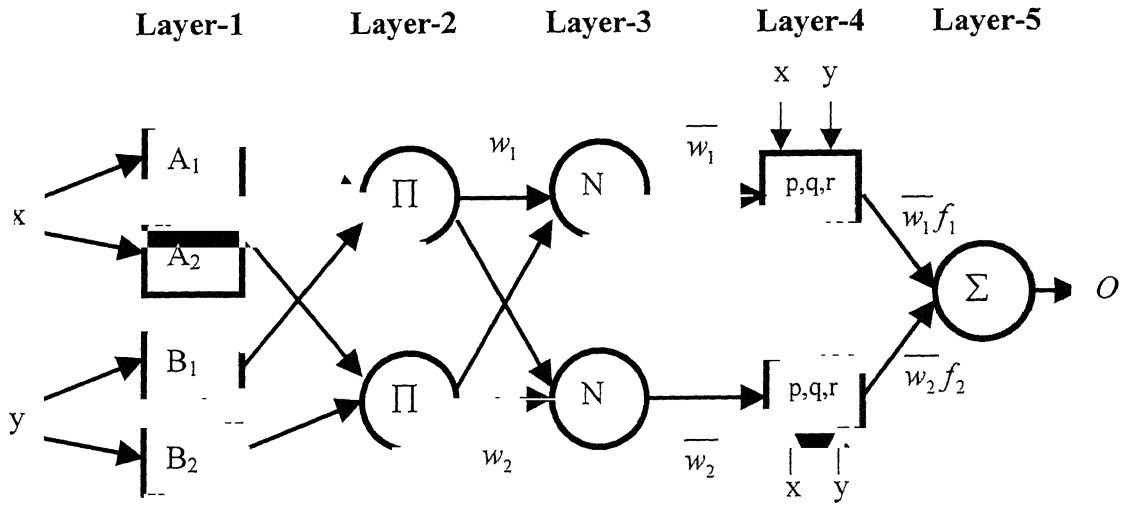


Fig. 3.1 ANFIS Architecture

The node functions in different layers are described below:

Layer-1: Every node i in this layer is a square node with node function

$$O_i^1(x) = \frac{1}{1+\left[\left(\frac{x-c_i}{a_i}\right)^2 b_i\right]} \qquad \ldots(3.4)$$

where x is the input to node i and $\{a_i, b_i, c_i\}$ is the node parameter set. Parameters in this node are referred to as premise parameters.

Layer-2: Every node in this layer is a circle node, which multiplies the incoming signals and sends the product out.

$$w_i = O_i^1(x) \times O_i^1(y) \qquad \ldots(3.5)$$

Each node output represents the firing strength of a rule.

Layer-3: The nodes in this layer normalize the outputs of all layer 2 outputs.

$$\overline{w_i} = \frac{w_i}{\sum_i w_i} \qquad \ldots(3.6)$$

Outputs of this layer are called as normalized firing strengths.

Layer-4. Every node in this layer is a square node with a node function

$$O_i^4 = \overline{w_i} f_i = \overline{w_i}(p_i x + q_i y + r_i) \qquad \ldots(3.7)$$

where p_i, q_i, r_i are node parameters. Parameters in this layer are referred to as consequent parameters.

Layer-5: This layer has the single node that computes the overall output

$$O_i^5 = \sum_i \overline{w_i} f_i \qquad \ldots(3.8)$$

3.3.3 Learning in ANFIS

- *Gradient Descent Technique*

Suppose that a given adaptive network has L layers and the k^{th} layer has #(k) nodes The node in the i^{th} position of the k^{th} layer is denoted by (k, i), and its node function by O_i^k. since a node output depends on its incoming signals and its parameter set, we have

$$O_i^k = O_i^k\left(O_1^{k-1},\ldots\ldots,O_{k-1}^{k-1},a,b,c,\ldots\ldots\right) \qquad \ldots(3.9)$$

where a, b, c etc., are the parameters pertaining to this node.

If the given training data set has P entries, we can define the error measure for the p^{th} entry of training data entry as the sum of squared errors:

$$E_p = \sum_{m=1}^{\#(L)}\left(T_{m,p} - O_{m,p}^L\right)^2 \qquad \ldots(3.10)$$

where $T_{m,p}$ is the m^{th} component of pth target output vector, and $O^L_{m,p}$ is the m^{th} component of actual output vector produced by the presentation of the pth input vector. Hence the overall error measure is $E = \sum_{p=1}^{P} E_p$.

In order to develop a learning procedure that implements gradient descent in E over the parameter space, first we have to calculate the error rate $\frac{\partial E_p}{\partial O}$ for p^{th} training data and for each node output O. The error rate for the output node at (L, i) can be calculated readily from (3.10).

$$\frac{\partial E_{PL}}{\partial O_{n,p}^L} = -2\left(T_{i,p} - O_{i,p}^L\right) \qquad \ldots(3.11)$$

For the internal node at (k, i), the error rate can be derived by the chain rule:

$$\frac{\partial E_p}{\partial O_{i,p}^k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E_p}{\partial O_{m,p}^{k+1}} \frac{\partial O_{m,p}^{k+1}}{\partial O_{i,p}^k} \qquad \ldots(3.12)$$

where $1 \le k \le L-1$. That is, the error rate of an internal node can be expressed as a linear combination of the error rates of nodes in the next layer. Therefore for all $1 \le k \le L$ and $1 \le i \le k$, we can find $\frac{\partial E_{pL}}{\partial O_{ii,p}^L}$ by (3 11) and (3 12)

Now if α is a parameter of the given adaptive network, we have

$$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \frac{\partial O^*}{\partial \alpha} \qquad \ldots(3.13)$$

where S is the set of nodes whose outputs depend on α. Then the derivative of the overall error measure E with respect to α is

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial \alpha} \qquad \ldots(3.14)$$

Accordingly, the update formula for the generic parameter α is

$$\Delta\alpha = -\eta \frac{\partial E}{\partial \alpha} \qquad \ldots(3.15)$$

in which η is the learning rate.

- *Least Squares Estimate Method*

  The network output can be expressed as:

$$output = F(\bar{I}, S) \qquad \ldots(3.16)$$

where $\bar{I}$ is the set of input variables and S is the set of parameters. If the parameter set S can be decomposed into two sets

$$S = S_1 \oplus S_2 \qquad \ldots(3.17)$$

(where $\oplus$ represents direct sum) such that output is linear in the elements of S_2, then given values of elements of S_1, we can plug P training data into (3.16) and obtain a matrix equation:

$$AX = B \qquad \ldots(3.18)$$

where X is an unknown vector whose elements are parameters in S_2. Let $|S_2| = M$, then the dimensions of A, X and B are P x M, M x 1 and P x 1, respectively. Since P (number of training data pairs) is usually greater than M (number of linear parameters), this is an over determined problem and generally there is no exact solution to (3.18). Instead, a least squares estimate (LSE) of X, X^*, is sought to minimize the squared error $\|AX - B\|^2$. The most well-known formula for X^* uses the pseudo-inverse of X,

$$X^* = \left(A^T A\right)^{-1} A^T B \qquad \ldots(3.19)$$

where A^T is the transpose of A, and $\left(A^T A\right)^{-1} A^T$ is the pseudo-inverse of A, if $A^T A$ is non-singular. While (3 19) is concise in notation, it is expensive in computation when dealing with the matrix inverse and it becomes ill defined if $A^T A$ is singular. As a result, we employ sequential formulas to compute the LSE of X. This sequential method of LSE is more efficient (especially when M is small) and can be easily modified to an on-line version for systems with changing characteristics. Specifically, let the i^{th} row vector of matrix A defined in (3.19) be a_i^T and the ith element of B be b_i^T, then X can be calculated iteratively using the sequential formulas widely adopted in the literature.

$$\begin{aligned} X_{i+1} &= X_i + S_{i+1} a_{i+1} \left(b_{i+1}^T - a_{i+1}^T X_i\right) \\ S_{i+1} &= S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}} \quad i = 0,1,2,\ldots\ldots,P-1 \end{aligned} \qquad \ldots(3.20)$$

where S_i is often called the covariance matrix and the least squares estimate X^* is equal to X_P. The initial conditions to bootstrap (3 20) are $X_0 = 0$ and $S_0 = \gamma I$, where γ is a positive large number and I is the identity matrix of dimension M x M.

- *Hybrid Learning Algorithm*

For the ANFIS architecture in Fig 3 1, given the values of the premise parameters, the overall output can be expressed, using (3.5-3.10), as linear combinations of the consequent parameters,

$$\begin{aligned} f &= \frac{w_1}{w_1 + w_2} f_1 + \frac{w_2}{w_1 + w_2} f_2 \\ &= \overline{w_1} f_1 + \overline{w_2} f_2 \\ &= (\overline{w_1} x) p_1 + (\overline{w_1} y) q_1 + r_1 + (\overline{w_2} x) p_2 + (\overline{w_2} y) q_2 + r_2 \end{aligned} \qquad \ldots(3.21)$$

This is linear in the consequent parameters (p_1, q_1, r_1, p_2, q_2 *and* r_2). As a result for ANFIS, from (3.17),

S = set of total parameters

S_1 = set of premise parameters

S_2 = set of consequent parameters

Therefore, the hybrid-learning algorithm presented above can be applied directly to ANFIS. The turn of events can be summarized as, in the forward pass of the hybrid algorithm, functional signals go forward till layer-4 and the consequent parameters are identified by the least squares estimate. In the backward pass, the error rates propagate backward and the premise parameters are updated by the gradient descent. However, the computational complexity of the least squares algorithm is higher than that of the gradient descent. For this reason, the gradient descent algorithm has also been implemented.

- *Heuristics used in Learning*

Usually, we can change the value of η to vary the speed of convergence. The heuristic rules for updating η are:

1. If the error measure undergoes eight consecutive reductions, increase η by 10%.

2. If the error measure undergoes an increment, decrease η by 50%

Though we can apply the gradient method to identify the parameters in an adaptive network, the method is generally slow and likely to become trapped in local minima. Here the hybrid-learning rule is used to identify parameters, which is a combination of gradient learning rule and least squares estimate.

3.3.4 Implementation of ANFIS

Depending on the number of membership functions associated with each input, the widths of membership functions are initialized in such a way as to spread across the whole interval [0, 1] The widths are all initialized to at 0.5 The training data to be presented to the network was normalized to the range 0.1 to 0.9 The outputs of the network are denormalized, in the end, to get the overall output. This kind of ANFIS is called Grid-Partitioned ANFIS, where the number of rules increases exponentially with the number of inputs. So when the number of inputs is large, ANFIS-GP can not be used effectively. Instead, a sub-clustered ANFIS is used, in which the number of rules is equal to the number of membership functions selected. The rules are initialized on the diagonal of the input space, with a higher membership function width at the center and gradually reducing widths on both sides of the diagonal.

3.4 Compensatory Neuro-Fuzzy System

3.4.1 Introduction

When the consequents in the fuzzy rules are fuzzy membership functions, the resulting neuro-fuzzy system is known as neuro-fuzzy system The overall output of the system is obtained using the centroid defuzzification. Optimization of the fuzzy membership functions will be done, by using learning algorithms to adjust the parameters of membership functions. The selection of optimal fuzzy operations and optimal reasoning mechanism was carried out by using a compensatory parameter, which can be updated during learning process, so that it automatically picks up the degree of compensation required between the pessimistic (min) and optimistic (max) operations. The neuro-fuzzy system incorporated with compensatory reasoning method is known as the Compensatory Neuro-Fuzzy System.

3.4.2 Universal Approximator using Compensatory Fuzzy Reasoning

The m fuzzy IF – THEN rules of the n input, one output compensatory fuzzy logic system are described below:

$$IF\ x_1\ is\ A_1^k\ and\ x_2\ is\ A_2^k\ and \ldots x_n\ is\ A_n^l\ THEN\ y\ is\ B^k \qquad .. (3.22)$$

Where A_i^k and B^k are fuzzy sets in the input and output space respectively, and are linguistic variables for i = 1,2...n and k = 1,2,3...m.

The fuzzy membership functions of A_i^k and B^k are of the form defined by (3.23),

$$\mu_{A_i^k}(x_i) = \exp\left[-\left(\frac{x_i - a_i^k}{\sigma_i^k}\right)^2\right] \qquad \ldots(3.23)$$

Let us consider,

$$\underline{x} = (x_1, x_2, \ldots\ldots, x_n)$$

For an input fuzzy set A', the kth fuzzy rule (3.22) will generate an output fuzzy set B'^k by using the sup-dot composition,

$$\mu_{B'^k}(y) = \sup\left[\mu_{A_1^k \times \ldots \times A_1^k \to B^k}(\underline{x}, y) \bullet \mu_{A'}(\underline{x})\right] \quad \ldots(3.24)$$

In (3.24), $\mu_{A_1^k \times \ldots \times A_1^k}$ is defined in a compensatory form as,

$$\mu_{A_1^k \times \ldots \times A_1^k}(\underline{x}) = (u^k)^{1-\gamma}(v)^{\gamma} \quad \ldots(3.25)$$

with a compensatory degree $\gamma \in [0,1]$

$$\begin{aligned} u^k &= \prod_{i=1}^{n} \mu_{i(x_i)}^k \\ u^k &= \left[\prod_{i=1}^{n} \mu_{i(x_i)}^k\right]^{1/n} \end{aligned} \quad \ldots(3.26)$$

from (3.25) and (3.26),

$$\mu_{A_1^k \times \ldots \times A_1^k}(\underline{x}) = \left[\prod_{i=1}^{n} \mu_{i(x_i)}^k\right]^{1-\gamma+\gamma/n} \quad \ldots(3.27)$$

Using the product-operation fuzzy implication and the dot-operation defined below,

$$\begin{aligned} \mu_A \to B(x,y) &= \mu_A(x)\mu_B(y) \\ \mu_A(x) \bullet \mu_B(y) &= \mu_A B(x,y) \end{aligned} \quad \ldots(3.28)$$

we obtain,

$$\mu_{B^k}(y) = \sup\left\{\mu_{B^k}(y)\mu_{A'}(\underline{x})\left[\prod_{i=1}^{n} \mu_{i(x_i)}^k\right]^{1-\gamma+\gamma/n}\right\} \quad \ldots(3.29)$$

A typical defuzzifier is defined by,

$$f(\underline{x}) = \frac{\sum_{k=1}^{m} b^k \delta^k \mu_{B^{ik}}(b^k)}{\sum_{k=1}^{m} \delta^k \mu_{B^{ik}}(b^k)} \qquad \ldots(3.30)$$

For a singleton fuzzifier, $\mu_{A^i}(\underline{x}) = 1$ and $\mu_{B^k}(b^k) = 1$

Therefore, from (3.29),

$$\mu_{B^k}(y) = \left[\prod_{i=1}^{n} \mu_{i(x_i)}^{k}\right]^{1-\gamma+\gamma/n} \qquad \ldots(3.31)$$

From (3.30) and (3.31), the compensatory neuro-fuzzy system output can be written as,

$$f(\underline{x}) = \frac{\sum_{k=1}^{m} b^k \delta^k \left[\prod_{i=1}^{n} \mu_{i(x_i)}^{k}\right]^{1-\gamma+\gamma/n}}{\sum_{k=1}^{m} \delta^k \left[\prod_{i=1}^{n} \mu_{i(x_i)}^{k}\right]^{1-\gamma+\gamma/n}} \qquad \ldots(3.32)$$

3.4.3 Architecture of Compensatory Neuro-Fuzzy System

The Compensatory Neuro-Fuzzy System has five functional layers. They are

- input layer,
- fuzzification layer,
- pessimistic-optimistic layer,
- compensatory operation layer, and
- defuzzification layer.

The general architecture of a compensatory neural fuzzy network is represented as in Fig. 3.2. A compensatory neuro-fuzzy system is initially constructed layer by layer according to linguistic variables, fuzzy IF-THEN rules, the pessimistic and optimistic

operations, the fuzzy reasoning method, and the defuzzification scheme of a fuzzy logic control system.

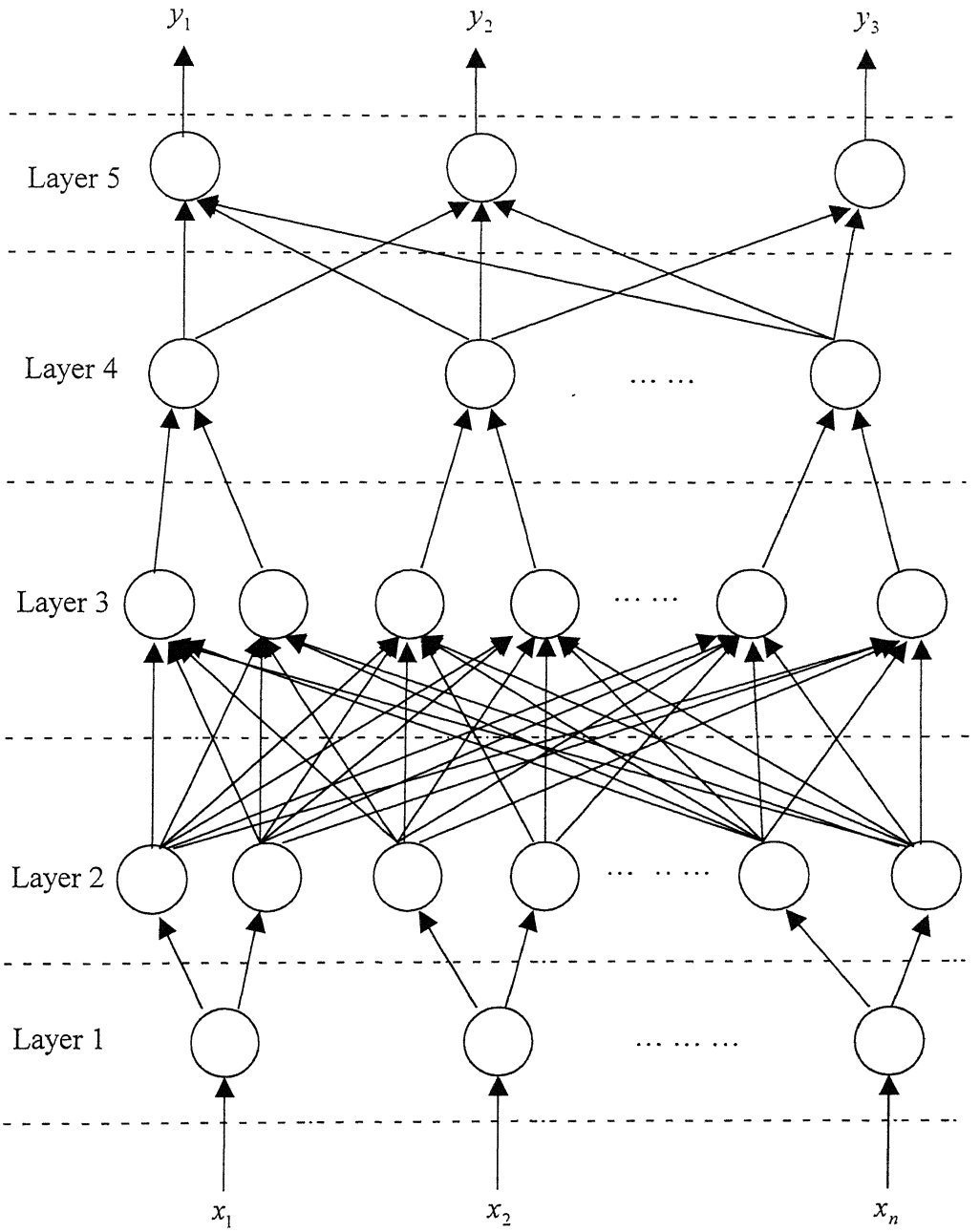


Fig. 3.2 CNFS Architecture

3.4.4 Fuzzy Neurons

There are various types of fuzzy neurons depending on various aspects. A few of them are listed below.

3.4.4.1 *Control Oriented Fuzzy Neurons*

- Fuzzification Neuron: The neuron, which performs the operation of fuzzification, is known as Fuzzification neuron. The fuzzification neuron performs a mapping between a crisp value x into a fuzzy set A_i, with degree y for $y=\mu_{Ai}(x)$, given by (3.23).
- Fuzzy reasoning Neuron: A simple fuzzy reasoning neuron can map the inputs x_i (i=1,2...n) to the output Y through some t-norm function $T(x_1, x_2, \ldots\ldots x_n)$. For example

$$T(x_1, x_2, \ldots\ldots x_n) = Min(x_1, x_2, \ldots\ldots x_n) \qquad \ldots(3.33)$$

  The complex fuzzy reasoning neuron can be constructed by using T-norm fuzzy neuron performing IF condition matching of fuzzy logic rules and a T-co norm fuzzy neuron integrating the fired rules.
- Defuzzification Neuron: A defuzzification neuron is one which generates a final crisp value based on the inputs x_i (I=1,2...n) and weights w_i The weights are the parameters of the membership functions.

$$D(x_1, x_2, \ldots x_n) = \frac{\sum_{i=1}^{n} w_i^1 w_i^2 x_i}{\sum_{i=1}^{n} w_i^2 x_i} \qquad \ldots(3.34)$$

where w_i^1 *and* w_i^2 are the centers and widths of the membership functions.

3.4 4.2 *Decision Oriented Fuzzy neurons*

Depending on the pessimistic and optimistic operations, the compensatory fuzzy neurons are defined

- Pessimistic Fuzzy Neuron: The pessimistic fuzzy Neuron can map the inputs x_i (i=1,2. n) to the pessimistic output by making a conservative decision for the pessimistic situation or even the worst case. Actually, the t-norm fuzzy neurons are pessimistic neurons. For example, here x_i (i=1,2...n) are in [0,1]

$$P(x_1, x_2, \ldots .. x_n) = Min(x_1, x_2, \ldots .. .. x_n)$$

$$P(x_1, x_2 ..... x_n) = \prod_{i=1}^{n} x_i$$

$$P(x_1, x_2 ..... x_n) = \frac{Min(x_1, x_2, ... x_n) + Max((x_1, x_2, ... x_n)}{10} \quad \ldots(3.35)$$

- Optimistic Fuzzy Neuron: On the other hand, the Optimistic Neuron can map the inputs x_i (I=1, 2.. n) to the optimistic output by making an optimistic decision for the optimistic situation or even for the best case. The t-co norm fuzzy neurons are optimistic neurons.

  For example, here, xi (I=1,2...n) are in [0,1]

$$O(x_1, x_2, ... x_n) = Max(x_1, x_2, ... x_n)$$

$$O(x_1, x_2 ..... x_n) = Max(x_1, x_2, ... x_n);$$

$$O(x_1, x_2 ..... x_n) = \frac{Min(x_1, x_2, ...x_n) + Max((x_1, x_2, ... x_n)}{2} \quad \ldots(3.36)$$

- Compensatory Fuzzy Neuron: The Compensatory Fuzzy Neuron can map the pessimistic input x_1 and optimistic input x_2 to the compensatory output C to make a relatively compromised decision for the situation between the worst case and the best

case. The bounded PI compensation has been utilized for the compensatory reasoning meachanism.

$$C(x_1, x_2) = x_1^{1-\gamma} x_2^{\gamma} \qquad ..(3.37)$$

where $\gamma \in [0,1]$ is called the compensatory degree.

The pessimistic and optimistic fuzzy neurons are very general fuzzy neurons used to make better decision based on the interval valued fuzzy sets. The compensatory fuzzy neuron can also perform the general operations of pessimistic and optimistic neurons.

3.4.5 Learning Algorithms of Compensatory Neuro-Fuzzy Systems

Given the n-dimensional input data vectors, $\mathbf{x}^p$, [$\mathbf{x}^p = (x_1^p, x_2^p \ldots x_n^p)$] and the one-dimensional output data vectors, $\mathbf{y}^p$, for p=1, 2...N, a supervised training algorithm has been used to optimally adjust the centers and widths of both the input and output membership functions of the neuro-fuzzy systems.

From (3.23), the fuzzy membership functions for an input vector, $\mathbf{x}^p$, [$\mathbf{x}^p = (x_1^p, x_2^p \ldots x_n^p)$] and the one-dimensional output data vectors, $\mathbf{y}^p$, for p=1, 2...N, are given by,

$$\mu_{A_i^k}(x_i^p) = \exp\left[-\left(\frac{x_i^p - a_i^k}{\sigma_i^k}\right)^2\right]$$
$$\mu_{B_i^k}(x_i^p) = \exp\left[-\left(\frac{x_i^p - b_i^k}{\delta_i^k}\right)^2\right] \qquad ...(3.38)$$

From (3.32), we have,

$$f(\underline{x}) = \frac{\sum_{k=1}^{m} b^k \delta^k z^k}{\sum_{k=1}^{m} \delta^k z^k} \qquad ...(3.39)$$

where,

$$z^k = \left[\prod_{i=1}^{n} \mu_{A_i^k}\left(x_i^p\right)\right]^{1-\gamma+\gamma/n}$$

The objective function can be defined as,

$$E^p = \frac{1}{2}\left[f\left(x^p\right) - y^p\right]^2 \qquad \ldots(3.40)$$

3.4.5.1 *Gradient Descent Algorithm*

According to the gradient descent method, the first partial derivative of the error function which is represented as the objective function, to be minimized, with respect to a variable is used to update the corresponding parameter of the neuro-fuzzy system. A learning constant, η, is used to multiply the gradient and the product is subtracted from the variable. The first partial derivatives with respect to different parameter variables and the corresponding corrections are given as follows:

1. Training the Centres of output membership functions:

$$b^k(t+1) = b^k(t) - \eta \left.\frac{\partial E^p}{\partial b^k}\right|_t$$

$$= b^k(t) - \eta \left.\frac{\left[f\left(x^p\right) - y^p\right]\delta^k z^k}{\sum_{k=1}^{m} \delta^k z^k}\right|_t \qquad \ldots(3.41)$$

2. Training the widths of the output Membership Functions:

$$\delta^k(t+1) = \delta^k(t) - \eta \left.\frac{\partial E^p}{\partial \delta^k}\right|_t$$

$$= \delta^k(t) - \eta \left.\frac{\left[f\left(x^p\right) - y^p\right]\left[b^k - f\left(x^p\right)\right]z^k}{\sum_{k=1}^{m} \delta^k z^k}\right|_t \qquad \ldots(3.42)$$

3. Training the centers of input membership functions:

$$a_i^k(t+1) = a_i^k - \eta \left.\frac{\partial E^p}{\partial a_i^k}\right|_t$$

where, ...(3.43)

$$\frac{\partial E^p}{\partial a_i^k} = \frac{2\left[f(x^p) - y^p\right]\left[b^k - f(x^p)\right]\left[x_i^p - a_i^k\right]\left[1-\gamma+\frac{\gamma}{n}\right]\delta^k z^k}{\sum_{k=1}^{m}\delta^k z^k}$$

4. Training the widths of the input Membership Functions:

$$\sigma_i^k(t+1) = \sigma_i^k - \eta \left.\frac{\partial E^p}{\partial \sigma_i^k}\right|_t$$

where, ...(3.44)

$$\frac{\partial E^p}{\partial \sigma_i^k} = \frac{2\left[f(x^p) - y^p\right]\left[b^k - f(x^p)\right]\left[x_i^p - a_i^k\right]\left[1-\gamma+\frac{\gamma}{n}\right]\delta^k z^k}{\sigma_i^{k3}\sum_{k=1}^{m}\delta^k z^k}$$

5. Training the Compensatory degrees: The compensatory factor γ was defined such that it could take values between the interval, [0,1], i.e., $\gamma \in [0,1]$. To incorporate this constraint inherently, γ can be redefined as follows,

$$\gamma = \frac{c^2}{c^2 + d^2} \qquad \ldots(3.45)$$

Then, for any value of c and d, $\gamma \in [0,1]$ will be satisfied. Thus, the external constraint on γ can be dispensed with. For updating the value of γ in each iteration, the following relations are used:

$$\frac{\partial E^p}{\partial \gamma} = \frac{\left[f(x^p) - y^p\right]\left[b^k - f(x^p)\right]\left[\frac{1}{n}-1\right]\delta^k z^k \ln\left[\prod_{i=1}^{n}\mu_{i(x_i)}^k\right]}{\sum_{k=1}^{m}\delta^k z^k} \qquad \ldots(3\ 46)$$

Once the partial derivative of the objective function with respect to the compensatory factor, γ is determined, its value can be used to update the values of c and d; and the value of γ will be updated in each iteration, as follows:

$$c(t+1) = c(t) - \eta \left\{ \frac{2\, c(t) d^2(t)}{\left[c^2(t) + d^2(t)\right]^2} \right\} \left| \frac{\partial E^p}{\partial \gamma} \right|_t$$

$$d(t+1) = d(t) + \eta \left\{ \frac{2\, c^2(t) d(t)}{\left[c^2(t) + d^2(t)\right]^2} \right\} \left| \frac{\partial E^p}{\partial \gamma} \right|_t \quad \ldots(3.47)$$

$$\gamma(t+1) = \frac{c^2(t+1)}{\left[c^2(t+1) + d^2(t+1)\right]}$$

In these formulae, t = 0, 1, 2...

3.4.6 Building an Initial Neuro-Fuzzy System

The neuro-fuzzy systems can be initialized by a heuristic algorithm to train the system more quickly Consider as many rules as the number of training patterns. Initialize the widths of the input and output membership functions equal to the values at the training patterns. The width can be initialized to 0.5. Lesser values of width increase speed of convergence but may cause the system to over fit. Each rule approximates the considered system at one training pattern. Fuzzy reasoning makes the output a continuous function. Fine-tuning of the membership function parameters is done to make the system approximate the given input data to the specified accuracy level. The compensatory reasoning mechanism presented above has been extended to ANFIS also, forming ANFIS-Compensatory.

3.5 Shortcomings of ANFIS and CNFS

- **Shortcomings of ANFIS**

Since the consequent parts of the rules are linear in the input variables, the total number of rules required may be higher. The number of rules in ANFIS is given by the formula (Number of labels to the power of Number of labels). Therefore, the only way to increase the number of rules is by increasing the number labels associated with each input. Then the number of rules increases exponentially with the number of inputs. One way to overcome this problem is instead of taking all the combinations of the input partitions, take only those along the diagonal (ANFIS-SC). Training them makes the input membership functions spread according to the data presented. In this case, the total number of rules becomes equal to the number of membership functions associated with each input. However, the initialization of the membership functions becomes tricky. Using the ANFIS-Co doesn't improve things either, because in that case the number of rules is uncontrollable, being equal to the number of training patterns.

- **Shortcomings of CNFS**

In case of CNFS, the number of rules is equal to the number of training patterns present in the input data. If the total number of training patterns is too high, so will be the number of rules, and the total number of parameters associated with the neuro-fuzzy system become too high unnecessarily. When training patterns are in close vicinity, the system will generate redundant rules, unnecessarily.

CHAPTER 4

GENERALIZED NEURO-FUZZY SYSTEMS

4.1 Introduction

Fuzzy systems can approximate any continuous function on a compact set to any accuracy [6]. However, it is difficult for even a domain expert, to examine all the input-output data coming from a complex system to determine the number of rules to implement the fuzzy system. The main issues associated with fuzzy systems are

1. parameter estimation, which involves determining the parameters of premises and consequences, and
2. structure identification, which concerns partitioning the input space and determining the number of fuzzy rules for a specific performance.

The problem of acquisition of fuzzy rules based on the learning ability of neural networks was addressed in the previous chapter, through ANFIS and CNFS, which are designed to approximate a process of fuzzy inference through the structure of neural networks. The main idea was the following: if some particular membership functions have been defined, to begin with a fixed number of rules by resorting to either trial and error method (ANFIS) or obtaining from the number of data patterns (CNFS) or by resorting to expert knowledge. Next, the parameters are modified by hybrid learning algorithm or back-propagation learning algorithm. These neuro-fuzzy systems readily solve two problems of conventional fuzzy reasoning:

1. lack of systematic design for membership functions and
2. lack of adaptability for possible changes in the reasoning environment.

These problems intrinsically concern parameter estimation. Nevertheless, structure identification, such as partitioning the input space and determination of number of fuzzy rules, is still time consuming. The problem of determining the number of fuzzy rules can be viewed as the choice of number of hidden nodes in neural. Two possible solutions for this problem can be.

1. Off-Line Identification: Determining the premise structure by clustering the input via an off-line clustering approach. As to the consequent part, only the value selected by clustering method is assigned to each rule initially. These parameters can be modified as learning progresses. The system so built can be called a static generalized neuro-fuzzy system.
2. On-Line Identification: To start the system with no rules and recruiting them dynamically, as proposed by ShiqianWu *et al.* [6], according to their significance to system performance so that not only can the parameters be adjusted but also the structure can be self-adaptive. The neuro-fuzzy system so built can be called dynamic generalized neuro-fuzzy system.

4.2 Static Generalized Neuro-Fuzzy Systems

Initially, the input data is clustered using a fuzzy rule-based clustering technique and then one rule is assigned for each cluster. The premise parameters are initialized at the cluster centre, and the consequent membership function centre being initialized at the highest value of output in that particular cluster.

4.2.1 Cluster Analysis and the ART1

Clustering refers to identifying the number of subclasses of c clusters in a data universe comprised of n data samples, and partitioning the data into c clusters. Note that, c=1 denotes that there are no clusters in the data whereas c=n constitutes the trivial case where each sample is in a "cluster" by itself. For numerical data, it can be assumed that the members of each cluster bear more mathematical similarity to each other than to the members of other clusters. One of the simple similarity measures is distance between pairs of feature vectors in the feature space. In many cases the number c of clusters in the data is not known. In such situations, it is necessary to identify the value of c that gives the most plausible number of clusters in the data Existing techniques like Adaptive Resonance Theory solve this problem by incorporating new clusters as and when required.

ART1 starts with no clusters. A vigilance parameter ρ is initialized which indicates the minimum degree of similarity that a data point should have with the cluster centre, to belong to that cluster. Whenever a data point comes, its degrees of similarities (according to some similarity chosen) are computed. If the maximum degree is less than ρ, then a new cluster should be considered. Whereas if the maximum degree of similarity is greater than ρ, then the data pointed is assigned to that cluster and the cluster centre is moved slightly towards the data point. When all the data points are presented, the total number of clusters gets decided and further presentation of data points fixes the cluster centres

4.2.2 Fuzzy Rule-Based Clustering Technique

Instead of the above mentioned clustering technique, in order to cluster the input data to the neuro-fuzzy system, rules can be used to do the same. Rule premises can be assumed as cluster centres, since the rule-firing strength in it-self represents inverse of the distance of the data point from the rule premise part. The rules used for clustering will be of the form,

$$IF\ x_1\ is\ A_1^k\ and\ x_2\ is\ A_2^k \cdots\ x_n\ is\ A_n^k\ THEN\ class\ is\ C_k \qquad \ldots(4.1)$$

where,

x_i are the inputs and

A_i^k are the membership function labels.

When a data points comes, all the rule firing strengths are computed. When the maximum firing strength below a vigilance threshold (ρ), a new rule will be generated, signifying a new cluster. However, if the maximum firing strength is higher than ρ, that data point can be assigned to the cluster for which the firing strength is the maximum, and that cluster centre can be moved slightly towards the newly added data point.

The problem associated with ART 1 and the rule-based clustering techniques is that the choice of vigilance threshold becomes tricky. Different values of ρ give different number of clusters for the same data. Some heuristics are needed to overcome this problem and to cluster the data into the most reasonable number of clusters. One heuristic used in this thesis is to compute the energy associated with the clustered data system and to find a vigilance threshold to minimize the energy. The energy function defined was: Sum of all the intra-cluster distances (+1/Minimum of the inter-cluster distance. Assuming the number of clusters to be c, the system energy is given by,

$$E = \sum_{ii=1}^{c} E_{ii} + \frac{1}{\min(E_{ij})} \quad \ldots(4.2)$$

where,

E_{ii} is the intra-cluster distance, of cluster i,

E_{ij} is the distance between clusters i and j.

Where intra-cluster energy is defined as the sum of distances, form all the points in a cluster to the cluster centre. The prime objective of defining the energy function is to find the vigilance threshold for which the intra-cluster distances are minimized and to the inter-cluster distances are maximized.

4.2.2.1 The complete algorithm

1. Initialize the vigilance parameter, and its increment.

$$\rho = 0$$
$$inc = 0 \quad \ldots(4.3)$$

2. When the first data point arrives, initialize a rule there. This involves initializing the centers of the membership functions at the data point and selecting a suitable width. The membership function being defined as,

$$\mu_{A_i^k}(x_i) = \exp\left[-\left(\frac{x_i - a_i^k}{\sigma_i^k}\right)^2\right] \quad \ldots(4.4)$$

where,

$i = 1,2...n$, n being the number of inputs,

$k = 1,2...c$, c being the number of rules,

A_i^k is the membership label of the ith input, in kth rule,

x_i is the input,

a_i^k is the center of the membership function,

σ_i^k is the width of the membership function

3. For the next data point onwards, calculate the firing strengths for all the rules. and select the rule with maximum firing strength. Firing strength being defined as

$$fs_k = \left[\prod_{i=1}^{n} \mu_i^k (x_i) \right] \quad \ldots(4.5)$$

4. If the maximum firing strength is greater than the vigilance threshold, the data point belongs to this cluster and so the centers of the rule membership functions are moved towards the data point

$$w_i^{new} = w_i^{old} + \eta \left(w_i^{old} - x_i \right) \quad \ldots(4.6)$$

5. Else, create a new rule and initialize the rule at the data point.
6. If more data points are present, go to step 3.
7. If the number of clusters remains same, go to step 2.
8. Compute the clustered system energy using (4.2).
9. If the system energy is less than previous computed value, increase the value of ρ ($\rho = \rho + inc$) and go to step 2.
10. Else, decrease ρ, ($\rho = \rho - inc$) and

    decrease increment-step ($inc = inc/10$), and go to step 2.

11. Repeat the above process until the energy becomes constant, or $inc < 0.001$.

The same algorithm can be used to find c clusters, with c being specified also. In this case instead of the energy, the number of clusters, c is utilized, to generate new clusters. Once the required number of clusters is formed, cluster centers can be learned.

4.2.3 Static Generalized ANFIS

The input data is clustered using the proposed fuzzy rule-based clustering technique, to determine the number of rules. The premises of the rules are initialized at the cluster centres. The consequents are initialized to zero. The parameters are then tuned using learning techniques of ANFIS.

4.2.4 Static Generalized CNFS

With CNFS also, the number of rules are decided using the fuzzy rule-based clustering algorithm, and premises are initialized at the cluster centres. However, the initialization of consequents is done in a heuristic way. In the cluster, the data point with the highest output is selected and that value of the output is selected for the membership function's centre. This can be extended to the case with number of the outputs greater than one, by selecting the maximum corresponding to one output.

4.3 Dynamic Generalized Neuro-Fuzzy Systems

A hierarchical on-line self-organizing learning is adopted so that the structure and parameter identification are done automatically and simultaneously without partitioning the input space and selecting initial parameters a priori. The system starts with no rules. Then, rules can be added dynamically according to their significance to system performance. The neuro-fuzzy system is constructed rule by rule i.e., each rule is generated according to criteria of rule generation.

4.3.1 Criteria of Rule Generation

4.3.1.1 *System Errors*

Output error of the system with respect to the teaching signals is an important factor in determining whether a new rule should be recruited. The error criterion can be described as follows

For each observation $(X^k, t^k), k = 1,2,...,n,$ where n is the number of total training data, X^k is the kth input vector and t^k is the kth-desired output, the overall GD-NFS output O^k will be computed. Define system error as

$$\|e^k\| = \|t^k - O^k\| \qquad \ldots(4.7)$$

If $\|e^k\| > k_e$, a new rule should be considered. Here, k_e is a predefined threshold, which may be kept constant or allowed to decay during the learning process. The idea behind k_e is to first find the most troublesome positions that have large errors and are not properly covered by existing rules. This idea is called coarse learning [5].

4.3.1.2 ε *-Completeness of Fuzzy Rules*

Definition: For any input in the operating range, there exists at least one fuzzy rule so that the matching degree (firing strength) is no less than ε. From the viewpoint of fuzzy rules, a fuzzy rule is a local representation over a region defined in the input space. If a new pattern satisfies ε-completeness, the GD-NFS will not generate a new rule but will accommodate the new sample by updating the parameters of the existing rules. The value of ε considered was 0.5.

4.3.1.3 *Training Process*

The training error can get stagnated based on two conditions. Either when the system is trapped in local minimum or when the system reaches its global minimum, i.e. the number of rules in the system is not enough for the problem under consideration. It is difficult to distinguish these two minima; however training is allowed to continue for some number of epochs (*ne*), to allow the system to come out of local minima, if it is trapped in one. If it doesn't improve the performance index, addition of a new rule should be considered. The new rule is initialized at a training pattern for which error is the maximum among all the training patterns. Once a new rule is added, the number of epochs (*ne*), for which training is continued, without considering addition of a new rule, is doubled.

4.3.2 Implementation of DG-NFS

In each epoch, all the criteria for rule generation are checked. If new rules are formed, the parameters associated with them are also updated using the back propagation algorithm. The initialization of the parameters of the newly generated rule is done using the pattern for which one of the criteria is failed. The rule membership functions are initialized at the same values as the data pattern, but the width is initialized to a lower value as compared to the widths initialized for the previous rules. i.e., as learning progresses the initial width chosen was reduced.

Instead of starting with no rules, the system can start with the rules generated by the clustering algorithm. New rules can be added further, if the current rules are not sufficient for the specified performance. The same criteria as presented above were used for generation of new rules.

CHAPTER 5

RESULTS AND DISCUSSION

In the present work, some of the benchmark problems that were used by researches to validate their algorithms, have been used to validate the implemented algorithms. The problems include Exclusive-OR, 4-Bit Parity, and Two Spiral problems, which come into the category of classification problems. For function approximation problems, the Sin(x)*Sin(y), and a Three Input Nonlinear Function have been considered. For prediction problems, the Mackey-Glass Time Series Prediction problem has been considered. The simulation results of these problems on various algorithms discussed earlier are given in the following sections.

5.1 OR/AND Network

The OR/AND neural network has been used to solve XOR and 4-bit parity problems, which fall under the category of classification problems. It was also used to approximate the nonlinear function Sin(x)*Sin(y).

5.1.1 The Exclusive-OR Problem

This is a two input-one output problem. Output is zero when both the inputs are equal (0 or 1), and output is one when they are unequal. The complete data set of XOR problem is given in Appendix B. Different types of AND and OR operations that were presented in section 2.2 have been used to solve the XOR problem, and the ones that gave consistent results are presented below:

Table 5.1 The various AND and OR operations

Number	AND operation	OR operation
0	Product	Derived from Product
1	Average Sum	Derived from Average Sum
2	Derived form Probabilistic Sum	Probabilistic Sum
3	Frank t-norm	Frank t-co norm
4	Hamacher t-norm	Hamacher t-co norm

The architecture that was used is (2-1), i e., two neurons in the input layer, and one output layer neuron. The error tolerance is 0.0001. Since the input layer neurons doesn't have any parameters associated the total number of parameters is equal to the number of parameters of the output layer neuron, which is 8. For ease in comparison, all the networks that are used for training are initialized at the same weights.

Table 5.2 Comparison of epochs for different AND and OR combinations (XOR problem)

Sl. No.	AND	OR	Epochs
1	0	0	483
2	0	1	823
3	0	2	483
4	0	3	488
5	0	4	387
6	1	2	833
7	1	3	739
8	1	4	627
9	2	2	483
10	2	3	487
11	2	4	382

The same problem was also solved with different types of aggregation functions implemented. The weighted mean was also used to solve the XOR problem. The network architecture in this case was (2-2-1). The total number of parameters is 24. The total number of training epochs is 3,712.

Table 5.3 Comparison of epochs with different Aggregations (XOR problem)

Sl. No.	Aggregation	Epochs
1	OR	483
2	Sigma Compensation	1117
3	Pi Compensation	746

5.1.2 4-Bit Parity Problem

In this work 4-bit parity problem has been considered. If the input pattern contains an even number of 1s, then its parity is 0 else it is 1. This is considered to be a difficult problem, because the patterns that are closer (using Euclidian distance) in the measure space, i.e. numbers that differ in only one bit require their answers to be different.

The network architecture used to train the data is (2-1). The total number of parameters associated is 12. The total number of training epochs the network took to get trained to an accuracy of 1e-12 is 54,290. Instead of using a single network for the whole input data, the data was divided into 4 parts, and 4 different networks of (2-1) are used to train the data. Error tolerance for each of the smaller network was 1e-12. Each of these networks on an average took 2,908 epochs to get trained to the accuracy of 1e-12. The total number of parameters for the 4 networks is 48. Although the number of parameters increased in the case of 4 networks, for the problem, the number of training epochs they took to get trained is remarkably reduced. When the data was divided into 4 parts, each of the sub-problems becomes equivalent to XOR problem, the solution

for which was presented earlier. When all the 4 networks can be trained in parallel, reduction in total training time can be achieved The complete table of results is attached in Appendix C.

5.1.3 Sin(x)*Sin(y) problem

General function approximation problems have been used by different researchers to validate their network capabilities and learning algorithms etc A popular function used for approximation is the Sin(x)*Sin(y). From the grid points of the range $[0,2\Pi]\times[0,2\Pi]$ within the input space of the above equation, 121 training data pairs were obtained. 76 testing data pairs were obtained from the intervals.

The OR/AND architecture used was 2-12-8-1, and has 21 neurons, with 229 parameters in all. The AND and OR operation considered are the product and probabilistic sum respectively. No activation function was considered in this case. The network took 83,448 epochs for an accuracy of 0.0001 (average sum-squared error).

When the sigmoid activation function was placed after the OR neuron in each OR/AND neuron, a network of 2-8-8-1 also gave similar results. Both the parameters associated with the nonlinearity were subjected to changes during the learning of the neuron. This adds two more parameters for each of the neurons. This network took 68,259 epochs to get trained to the average error sum of 0.0001. But the networks gave different results on different runs because of the random initialization of weights. Some times the networks failed to converge.

The simulation results for the network (2-12-8-1), without activation function are presented below:

OR/AND network Architecture: (2-12-8-1); minimum error achieved: 0.0001.

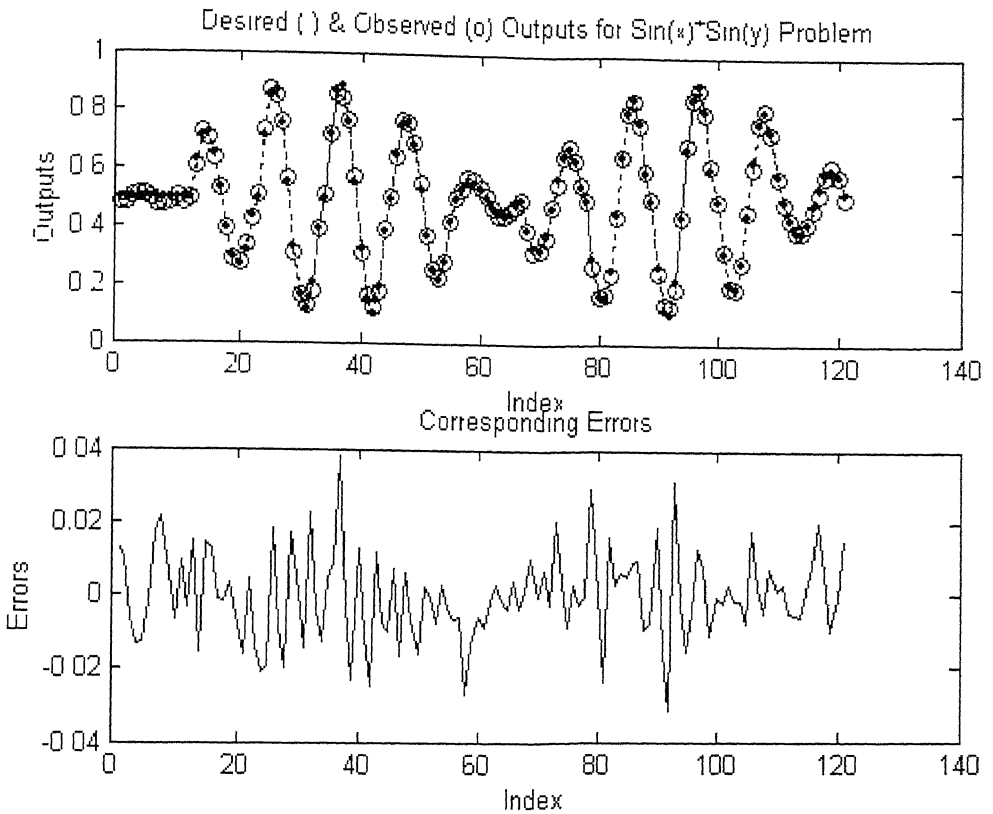


Fig. 5.1 Desired (.) & Observed (o) outputs for Sin(x)*Sin(y) Problem (2-12-8-1)

From the above figure it is evident that the maximum errors are for the data points where output is large. Addition of more neurons may improve the results, but at the cost of increased number of parameters and slow convergence.

In this case the data was divided into 16 smaller parts, and a smaller network (2-1) was used to approximate the function in each of the smaller region. The AND and OR operations considered for these networks are also the same as in the earlier case. No activation function was considered in these networks also. Each of the networks has only 6 parameters making the total parameters 96. Each of the smaller networks on an average took 4,999 epochs to get trained.

Simulation results for the case when 16 smaller networks (2-1) were used:

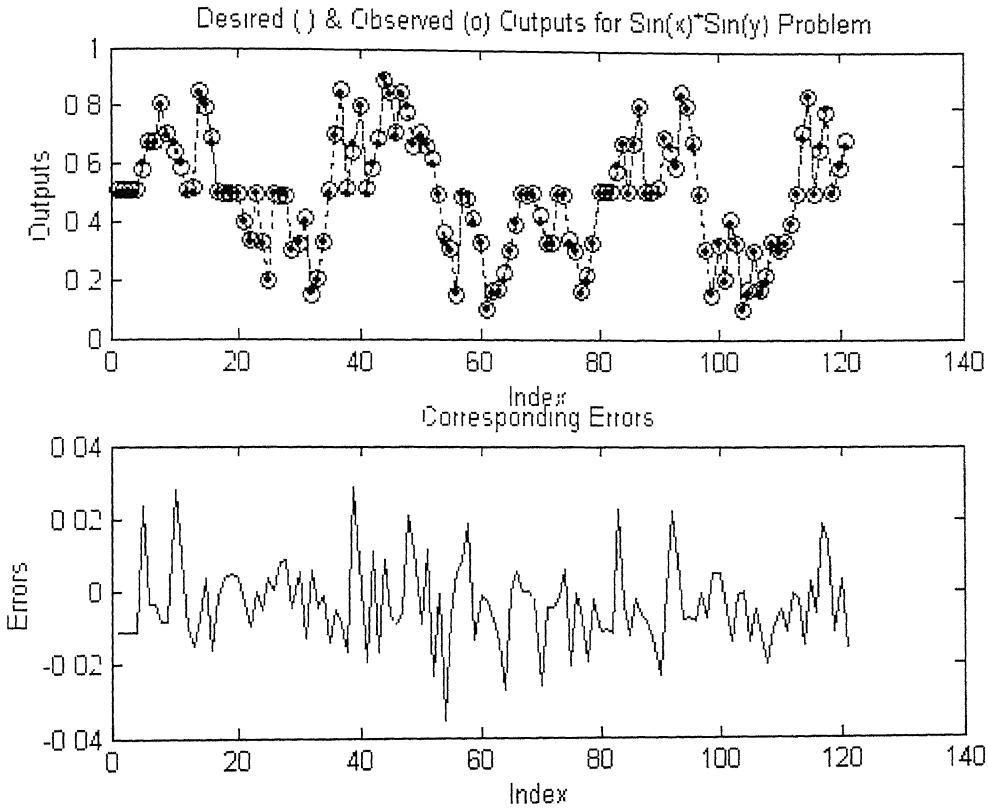


Fig. 5.2 Desired (.) & Observed (o) outputs for Sin(x)*Sin(y) Problem (16: 2-1)

The reduction in number of parameters is significant. Since the number of parameters is less learning time is also reduced. This clearly indicates the need for algorithms that can divide the data into smaller parts and have one smaller network approximate the function in each of the smaller regions.

Other problems that were mentioned in the beginning of the chapter, have also been attempted to solve, by OR/AND networks, but since the results are not consistent, they are not presented.

5.2 INITIAL NEURO-FUZZY SYSTEMS

5.2.1 ANFIS

The three different types of ANFIS that were developed are

- Grid-Partitioned (GP)
- Sub-Clustered (SC)
- Compensatory (Co)

5.2.1.1 XOR, 4-Parity and Sin(x)*Sin(y) Problems

Table 5.4 Simulation Results for XOR, 4-Parity and Sin(x)*Sin(y) Problems (ANFIS)

Problem	ANFIS used	Number of rules	Number of Epochs	Error reached
XOR	GP	4	589	1e-12
	GP	2	663	1e-12
	SC	2	671	1e-12
	Co	4	663	1e-12
4-Bit Parity	GP	16	600	1e-12
	SC	4	842	1e-12
	Co	16	824	1e-12
Sin(x)*Sin(y)	GP	8	99,999	0.0025
	GP	16	10,593	0.0001
	SC	8	99,999	0.00391
	SC	10	99,999	0.00215
	Co	121	80	0.0001

The error measure that was used in the simulations shown above is the sum-squared error. From the above results it is evident that increasing the number of rules and thereby the total number of parameters improves the performance of the system, contrary to artificial neural networks, where increasing the number of hidden layers or number of

neurons in hidden layers, makes the learning process slow, although it improves the generalization capabilities. For the Sin(x)*Sin(y) problem, by increasing the total number of rules to 121, which is equal to the number of training pairs, i.e. one rule for each of the training pair, makes the system learn in 80 epochs, which with 16 rules, took 10,593 epochs. The other advantage with the above systems and with neuro-fuzzy systems in general is the initialization, which is not random, and hence learning takes the same form on any number of runs.

5.2.1.2 The Two Spiral Problem (Proposed: Alexis Wieland of MITRE lrp. [2])

This is a benchmarking classification problem The problem consists of Np=198 (x,y) pairs of points that lie on interlocking spirals (Np/2 on each) separated in space by 180 degrees, that go around a common origin three times. Points on each spiral belongs to one class and the network has to associate points in the sample space with the correct class (0,1). This problem has been solved using ANFIS-GD and ANFIS-Co.

Table 5.5 Simulations Results for Two Spiral Problem (ANFIS)

Problem	ANFIS used	Number of rules	Number of Epochs	Error reached
Two Spiral Problem	GP	64	1205	1.76e-03
	Co	4	75	1.23e-03

Even in this case it is evident that more rules give better approximation and faster solution. The trade-off for faster learning is the increased number of parameters. Although the total number of epochs for the case with 194 rules is less the time taken for each epoch is more compared to the case with 64 rules. The error measure used was average sum-squared error.

Plots for ANFIS-GP; Number of rules: 64; Error tolerance 1.7e-03

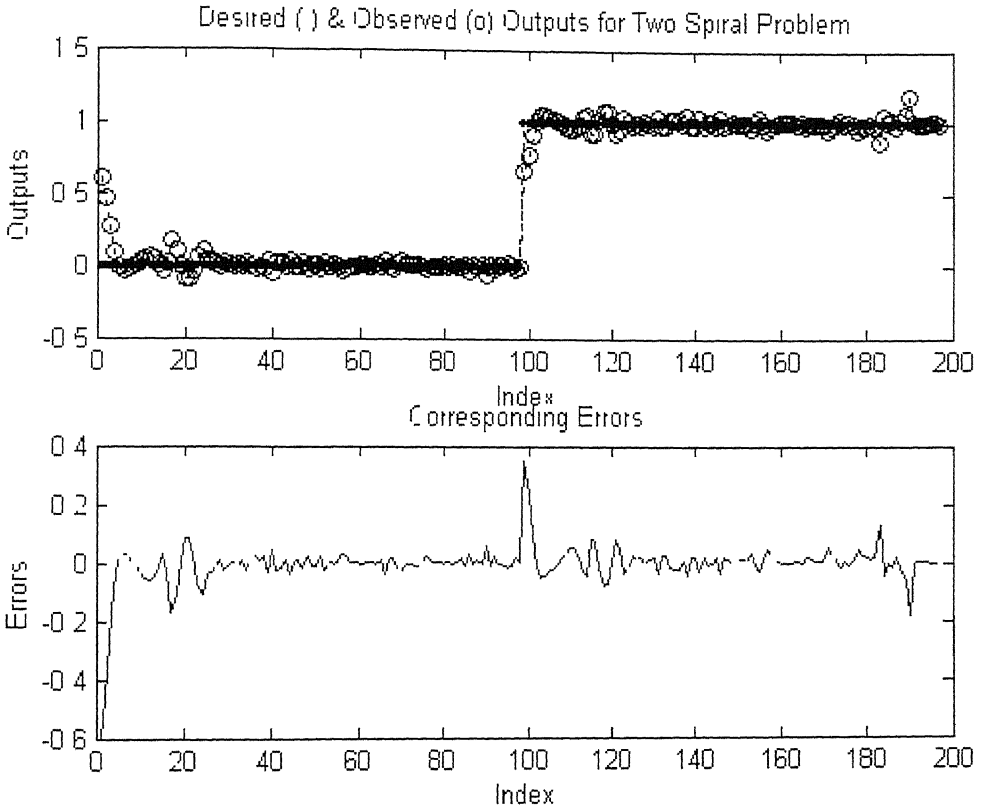


Fig. 5.3 Desired (.) and Observed (o) outputs for Two Spiral problem (ANFIS)

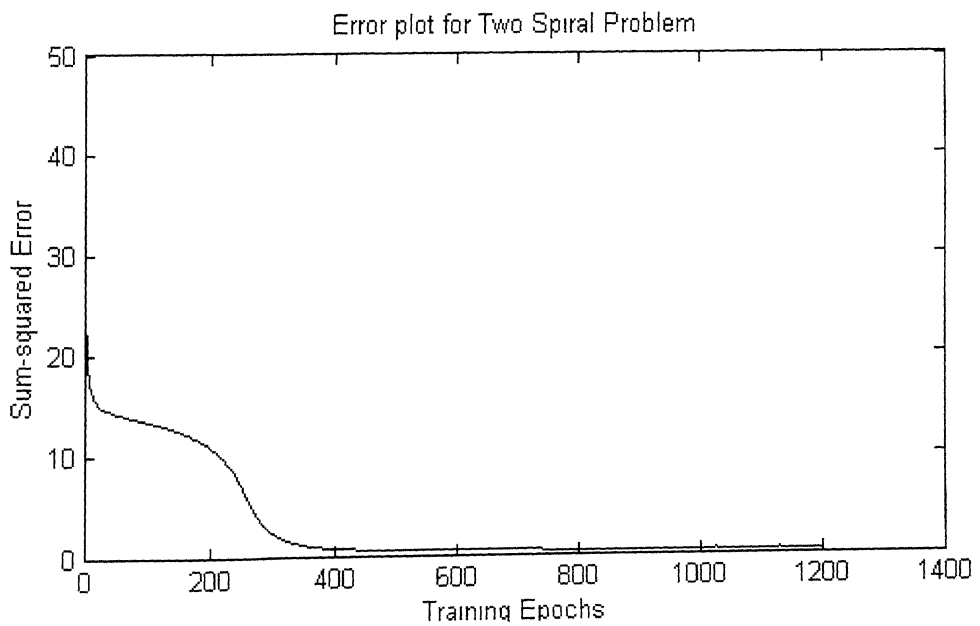


Fig. 5.4 Training error plot for Two Spiral problem (ANFIS)

5.2.1.3 Modeling a three input nonlinear function

The training data for this problem are obtained from

$$output = \left(1 + x^{0.5} + y^{-1} + z^{-1.5}\right)^2 \qquad \ldots(5.1)$$

216 training data and 125 checking data have been generated uniformly from the input ranges [1,6]*[1,6]*[1,6] and [1.5,5.5]*[1.5,5.5]*[1.5,5.5], respectively The training data was used to train the ANFIS, while the checking data was used for verifying the identified ANFIS only.

Table 5.6 Simulation Results for Three Input Nonlinear Function (ANFIS)

Problem	ANFIS used	Number of rules	Number of Epochs	Error reached
Three Input Nonlinear Function	GP	27	357	0.0001
	GP	8	9999	0.0279
	SC	8	99999	0.0037

As has been observed in the previous cases, increase in number of rules improves system generalization capability and learning speed, in this case also.

5.2.1.4 Predicting Chaotic Dynamics

In this problem, ANFIS has been employed to predict future values of a chaotic time series. The time series used in this simulation is generated by the chaotic Mackey-Glass differential delay equation [3] defined below,

$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1 + x^{10}(t-\tau)} - 0.1x(t) \qquad \ldots(5.2)$$

The prediction of future values of this time series is a benchmark problem, which has been considered by a number of researchers. The goal of the task is to use known values up to

the point x=t to predict the value at some point in the future x=t+P. The standard method for this type of prediction is to create a mapping from D points of the time series spaced *d* apart, that is, (x(t-(D-1)*d*),. .,x(t-*d*),x(t)), to a predicted future value x(t+P). The values D=3, D=4 and D=6 has been used in this simulation.

From the Mackey-Glass time series x(t), 1000 input-output data pairs have been extracted, of which the first 500 have been used for training and the rest for checking

Table 5 7 Simulation Results for Time Series Prediction Problem (ANFIS)

Problem	ANFIS used	Number of rules	Number of Epochs	Error reached
Mackey-Glass Time series Prediction (D=4)	GP	16	1160	0.00015
	SC	12	486	0.001
With (D=3)	GP	8	1054	0.0006

Considering the faster learning of ANFIS, it can be said that the ANFIS has captured the underlying dynamics. A disadvantage of ANFIS-GP has been surfaced in this case, where, 16 rules are found to be incapable of learning to an accuracy of 0.0001, so the need for increasing the number of rules being identified, it can only be done by increasing the number of membership functions associated with each input. When the membership functions are 2, the total number of rules is 16, but when they are increased to 3, the number of rules becomes 81, which is clearly way higher than what the system requires.

Plots for ANFIS-GP; number of rules: 16; Error reached: 0.00015.

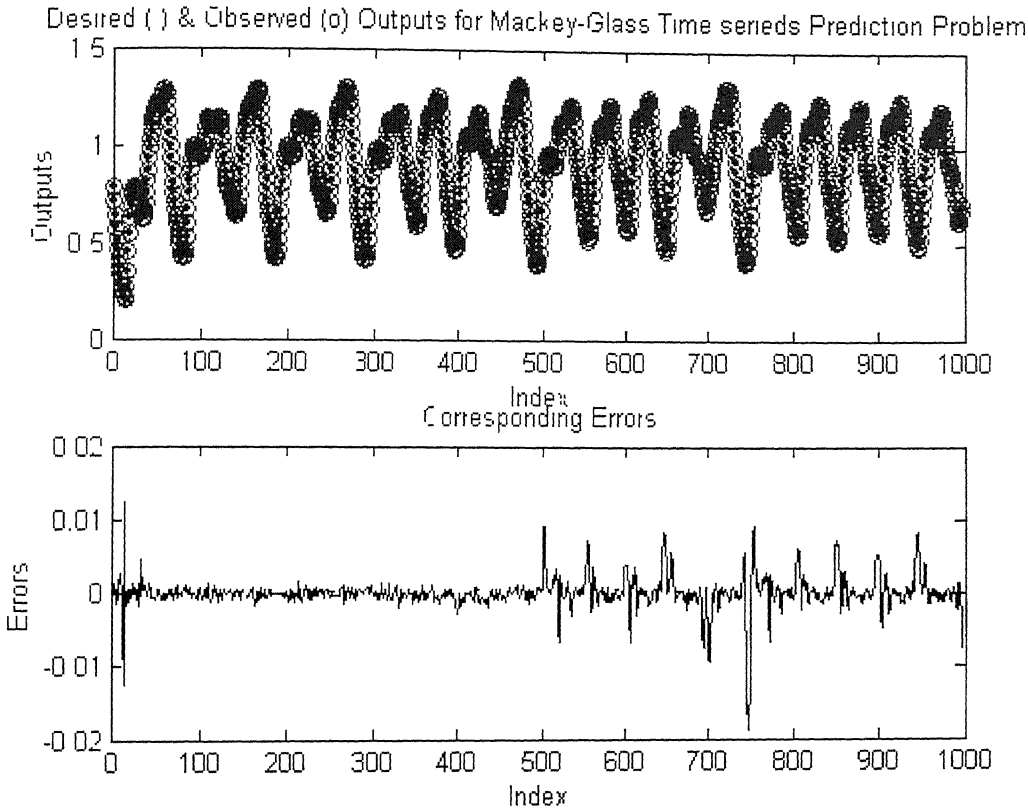


Fig. 5.5 Desired (.) and Observed (o) outputs for MGTSPP (ANFIS)

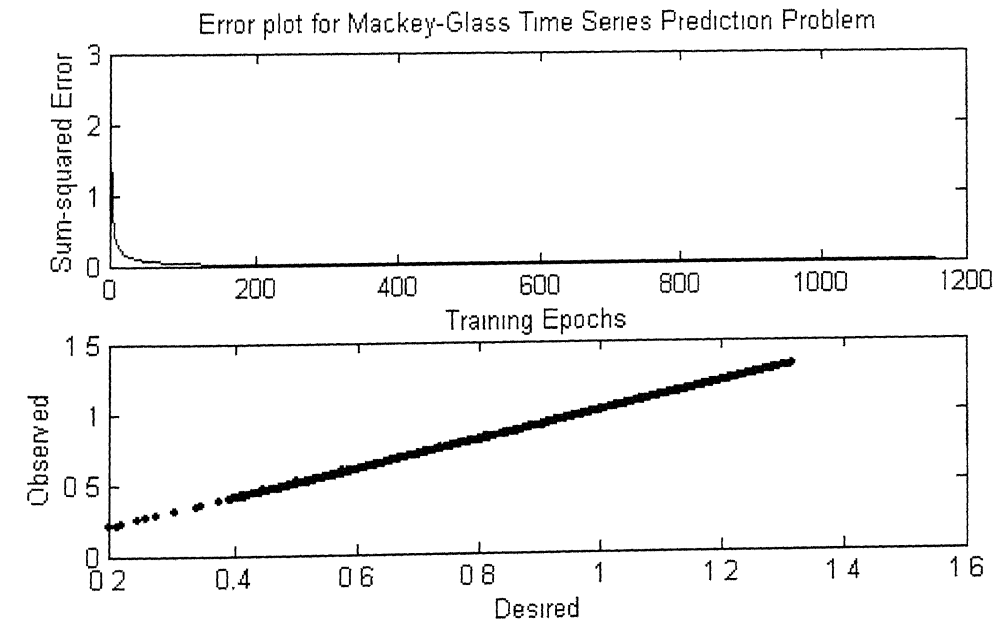


Fig. 5.6 Training errors and Desired vs Observed plots for MGTSPP (ANFIS)

In Fig 5.5, the first 500 points are the training patterns and the remaining are the checking patterns, which are not presented during the training. Although the checking errors are more compared to the training errors, they are in tolerable limits.

5.2.2 CNFS

All the above mentioned problems have also been solved by CNFS The results are presented below·

Table 5.8 Simulation Results for benchmarking problems (CNFS)

Problem	Number of rules	Number of Epochs	Error reached
XOR	4	103	1e-12
4-Bit Parity	16	164	1e-12
Sin(x)*Sin(y)	121	884	0.0025
Three Input Nonlinear Function	216	446	0.001
Two Spiral	198	1078	0.0629
Mackey-Glass Time series (D=3) Initial width=0.4	500	671	0.011
Mackey-Glass Time series (D=3) Initial width=0.05	500	671	0.00238
Mackey-Glass Time series (D=4) Initial width=0.4	500	705	0.0174
Mackey-Glass Time series (D=4) Initial width=0.05	500	705	0.00289

In CNFS, the number of rules is equal to the total number of training patterns present. So, the total number of parameters associated with the system can be too high. increasing the search space, and making the selection of initialization membership function width difficult. With smaller values, the network gets trained faster, which is evident from the results obtained for the Mackey-Glass Time Series Prediction problem.

5.2.2.1 Simulation Results for Two Spiral Problem

CNFS rules: 194, Initial membership function width: 0.1;

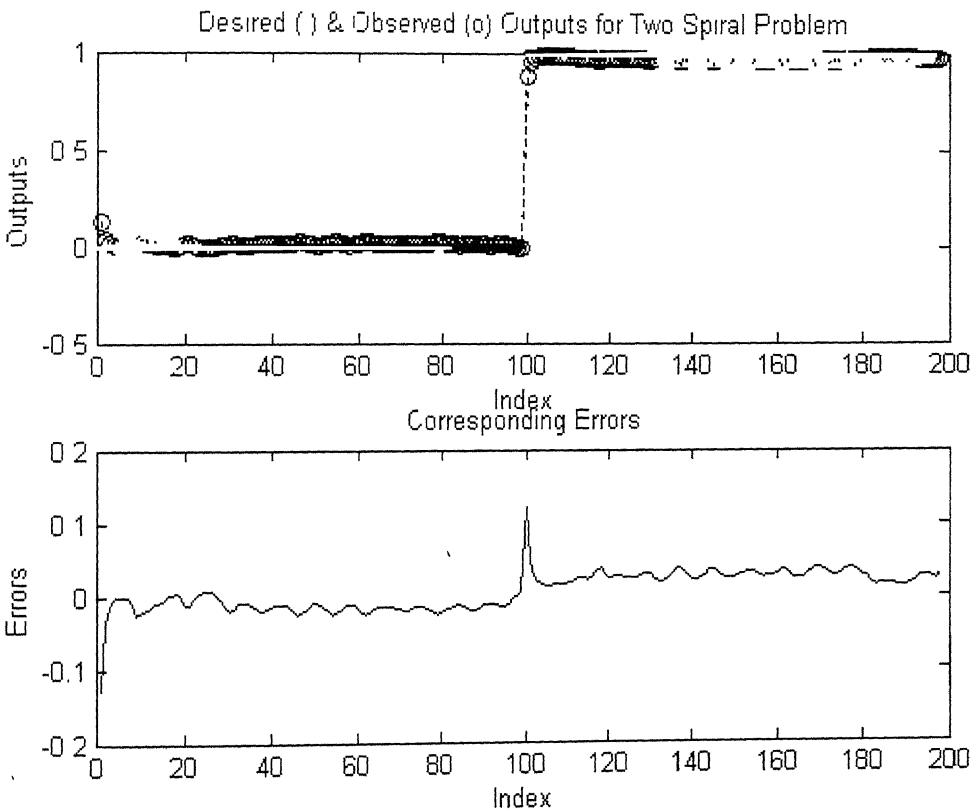


Fig. 5.7 Desired (.) and Observed (o) outputs for Two Spiral Problem (CNFS)

The results in Fig. 5.3 and Fig. 5.7 indicate that CNFS gave better results for Two Spiral Problem as compared to ANFIS. Also comparing the simulation results for various other problems, the CNFS seems to be giving better results for classification problems,

whereas convergence and performance index in ANFIS are better for function approximation problems.

5.3 Static Generalized NFS (SG-NFS)

In these algorithms, the input data is first clustered using the fuzzy rule-based clustering algorithm proposed, and one rule is initialized at each of these clusters. The performance of Static Generalized ANFIS and CNFS are presented below:

Table 5.9 Comparison of Simulation Results of SG-ANFIS and SG-CNFS

Problem	System	Number of rules	Number of Epochs	Error reached
XOR	ANFIS	4	391	1e-12
	CNFS	4	103	1e-12
4-Bit Parity	ANFIS	16	334	1e-12
	CNFS	16	164	1e-12
Sin(x)*Sin(y)	ANFIS	16	593	0.0028
	CNFS	16	12,459	0.009
	CNFS	8	99,999	0.0148
Three Input Nonlinear Function	ANFIS	27	313	0.0001
	CNFS	27	997	0 0034
Mackey-Glass Time series (D=6)	ANFIS	14	141	0 0065
	CNFS	14	601	0 0065
Mackey-Glass Time series (D=4)	ANFIS	11	414	0.00013
	CNFS	11	23,322	0.0019

The SG-ANFIS, gave a performance index of 0.00013 with only 11 rules, when ANFIS-GP failed to get to a performance index less than 0.00015 even with 16 rules (see Section 5.2.1.4). This can be attributed to the proper of partitioning of input space in case of SG-ANFIS.

5.3.1 Simulation results for Three Input Nonlinear Function Problem

SG-ANFIS: Number of rules: 27; Minimum error achieved 0.0001.

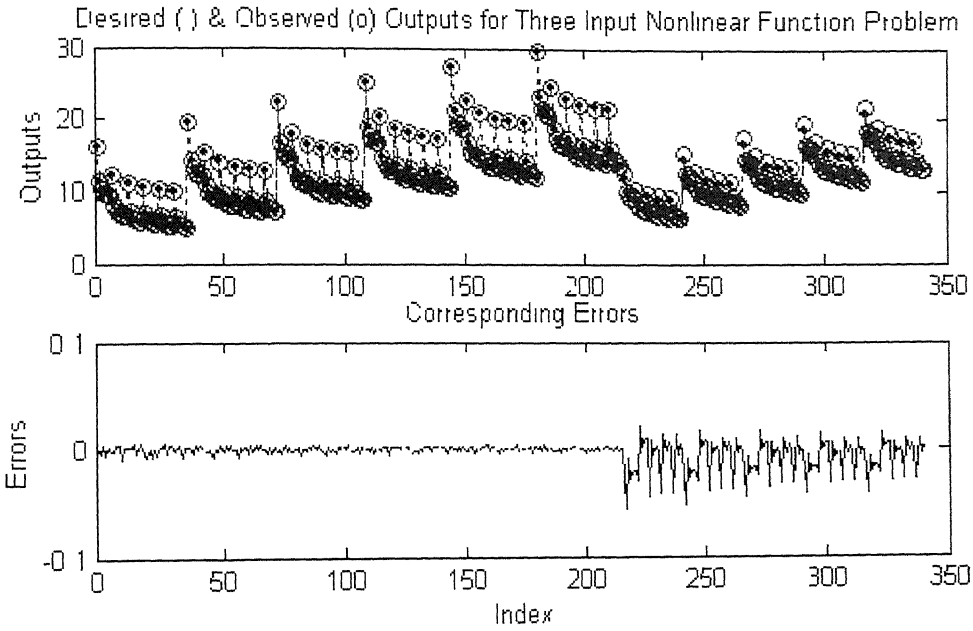


Fig 5.8 Desired ( ) and Observed (o) outputs for TINF Problem (SG-ANFIS)

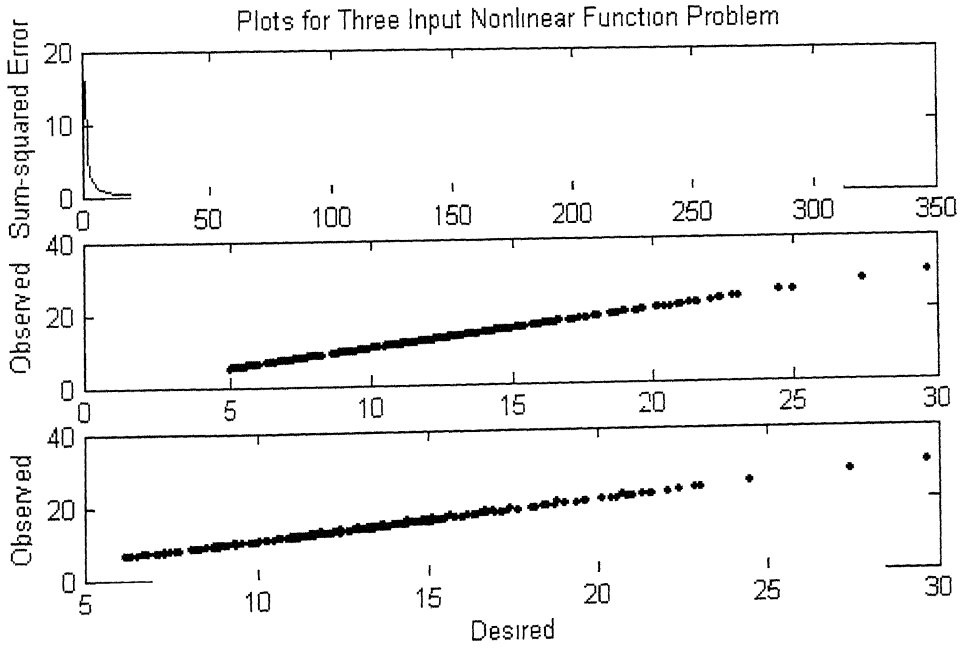


Fig. 5.9 Training errors and Desired vs. Observed plots for TINF problem (SG-ANFIS)

From the above results, it can be observed that as in the case of ANFIS and CNFS, between the SG-ANFIS and SG-CNFS, SG-ANFIS gave better performance, in lesser

number of epochs, for function approximation problems and SG-CNFS performed better in terms of number of epochs and accuracy for classification problems. This may be attributed to the difference in the type of consequents employed. For ANFIS, the consequent was linear in the input variables and hence ANFIS makes a linear approximation of the output in the respective regions covered by each of the rules. So learning in ANFIS basically means to identify the input spaces where the output can be approximated by a linear combination of the inputs. On the contrary, the consequent in CNFS is a bell-shaped membership function, and because of that, learning in CNFS means to identify the regions where function maxima are present, then cover those regions by the rules, and to place the output membership function at the maxima. In other words rules in CNFS tend to align themselves at the function maxima.

In classification problems, the outputs for two very close data points can vary between the maximum and the minimum. Hence they are highly nonlinear, having peaks and valleys alternately (undulations), or closely. Where as in function approximation, although the functions considered are nonlinear, the undulations may be lesser or the linear regions are more. This could be one of the reasons for the better performances of ANFIS and CNFS in function approximation and classification problems respectively.

5.4 Dynamic Generalized Neuro-Fuzzy Systems

In this algorithm, the system starts with zero rules and recruits them dynamically according to the rule generation criteria given in section 4.3.1. The simulation results for various problems are given below:

Table 5.10 Comparison of Simulation Results for DG-NFS and SG-NFS

Problem	NFS	Number of rules	Number of Epochs	Error reached
XOR	DG	3	139	4e-09
	SG	4	85	4e-09
4-Bit Parity	DG	9	468	1e-05
	SG	16	106	1e-05
Sin(x)*Sin(y)	DG	7	8,692	1.5e-04
	SG	16	5,605	1.5e-04
Three Input Nonlinear Function	DG	14	19,999	3.14e-04
	SG	27	267	3 14e-04
Mackey-Glass Time series (D=4) Prediction	DG	4	9,999	5.3e-05
	SG	11	9,652	5.3e-05

The errors shown in the above table are average sum squared errors. Although the errors obtained in DG-NFS are more, the reduction in number of rules and there by the total number of parameters is significant. Hence the trade-off for a less number of variables is a reduction in performance index achievable and an increase in the number of epochs.

5.4.1 Simulation results for 4-Bit parity Problem

The addition of rules that occurred during training of the DG-NFS for 4-Bit Parity Problem is presented below:

Table 5.11 The process of addition of rules for 4-Bit Parity Problem (DG-NFS)

Rules	1	2	3	4	5	6	7	8	9
Epoch	1	26	92	150	192	217	258	307	414
Error at addition	2 455	1.674	1.586	1.319	0.614	0.475	0.314	0.173	5e-05
Minimum Error Reached	1.297	1.110	0.947	0.674	0.576	0.442	0.264	2e-04	1e-05

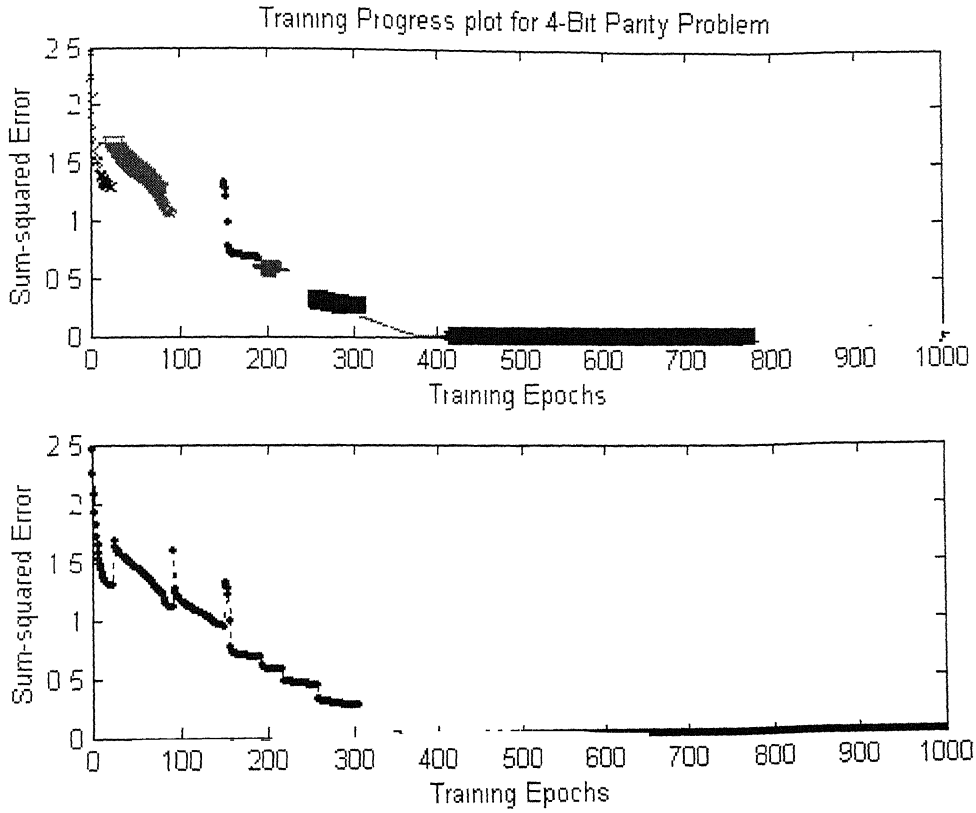


Fig. 5.10 Progress of training for 4-Bit Parity Problem (DG-NFS)

The increase in error, when a new rule is added, up to the fourth rule can be attributed to the reason that the number of rules in the system does not cover the whole input space. Because the rules added in the initial part of training cover a large region, all the outputs in the region get affected, causing an increase in error. However, from the fifth rule onwards, addition of a rule causes significant decrease in error, immediately. From the above figure, it is clear that after a rule being added, new rule is considered only when the decrease in error falls below a threshold, according to rule generation criterion 3, presented in section 4.3.1.

5.4.2 Simulation results for Time Series Prediction Problem (Mackey-Glass)

DG-NFS. Number of rules: 4; Minimum error achieved: 5.3e-05

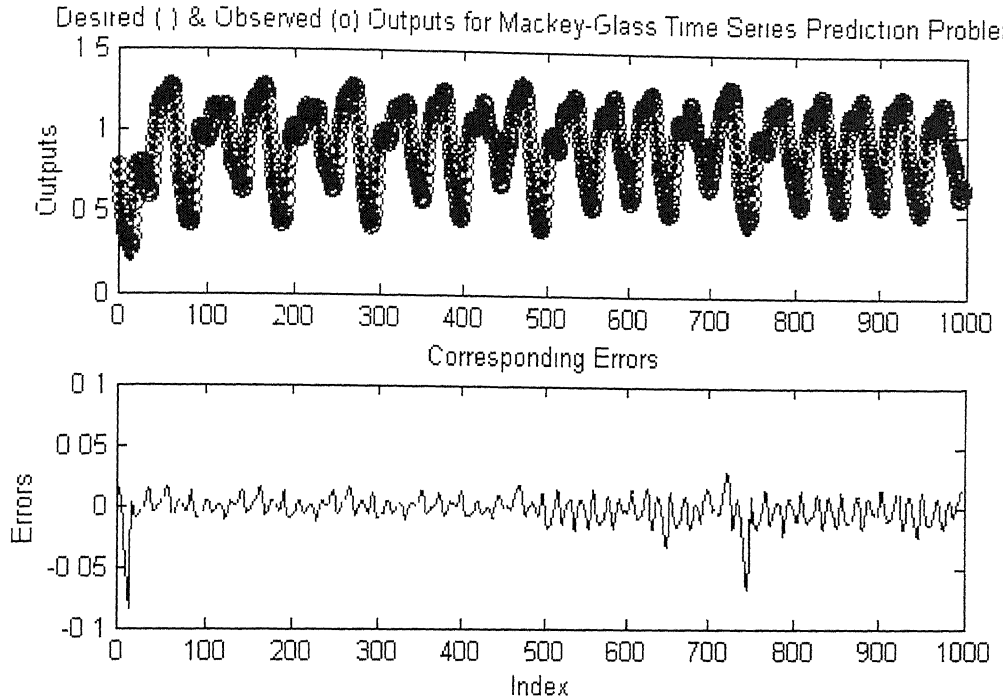


Fig. 5.11 Desired (.) and Observed (o) outputs for MGTSPP (DG-NFS)

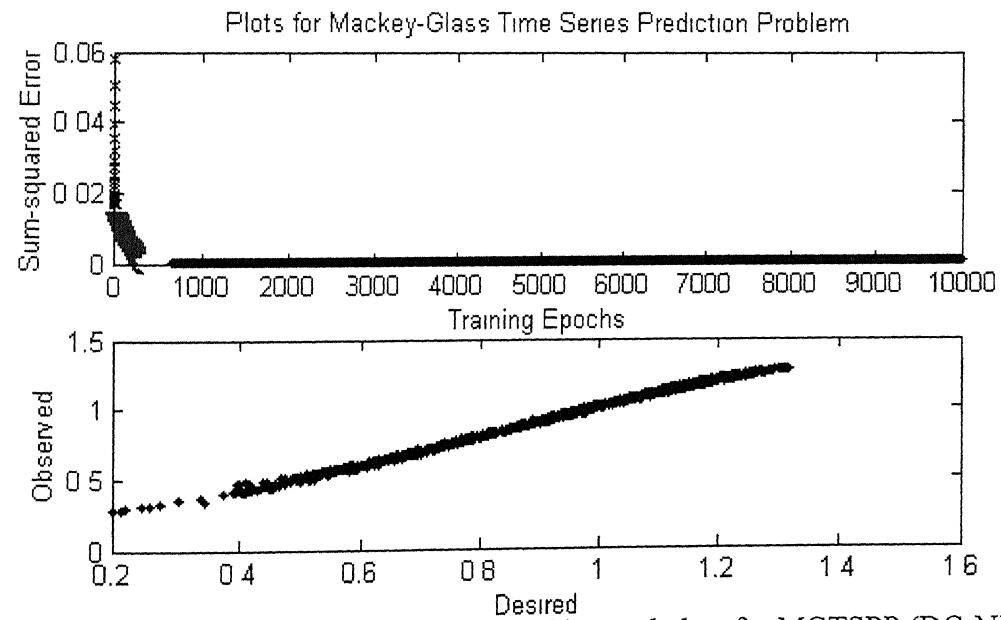


Fig. 5.12 Training progress and Desired vs. Observed plots for MGTSPP (DG-NFS)

Although the performance index at the end of training is not as small as in the case of ANFIS-GP, testing results are in close range, with the testing error being 7.2e-05. The

minimum average error sum achieved for the same problem with SG-ANFIS (with 11 rules) was 2 3e-07 (Table 5.8), where as the error achieved with DG-NFS (with 4 rules) is only 5.3e-05 (Table 5 9).

Two additional constraints that were put on addition of rules, in this case are

1. a new rule is added, only if the maximum of the errors of individual patterns is greater than a threshold The threshold taken in this case was 0.025.
2. Only one rule can be added for any particular pattern.

The errors for the pattern 16 was greater than the threshold mentioned above in constraint 1, but because of constraint 2, which prohibits addition of a second rule for a particular pattern, no more than 4 rules were added even when the decrease in error has become insignificant.

5.5 Fuzzy Rule-Based Clustering Algorithm

To validate the algorithm proposed, it was tested on a problem with data points from 4 ellipses, spaced considerable distance apart. The data is included in Appendix D. Some outliers were also present in the data. The algorithm was able to cluster the data correctly, though the cluster centers obtained were not optimum. The algorithm can also be used to obtain a pre-specified number of clusters. In this case it was used to generate 5 and 8 clusters. The training process for clustering the data to achieve minimum of the energy function defined in Section 4.2.2 is also included in Appendix D.

The plot showing the data and the cluster centers, obtained from the clustering algorithm are presented below:

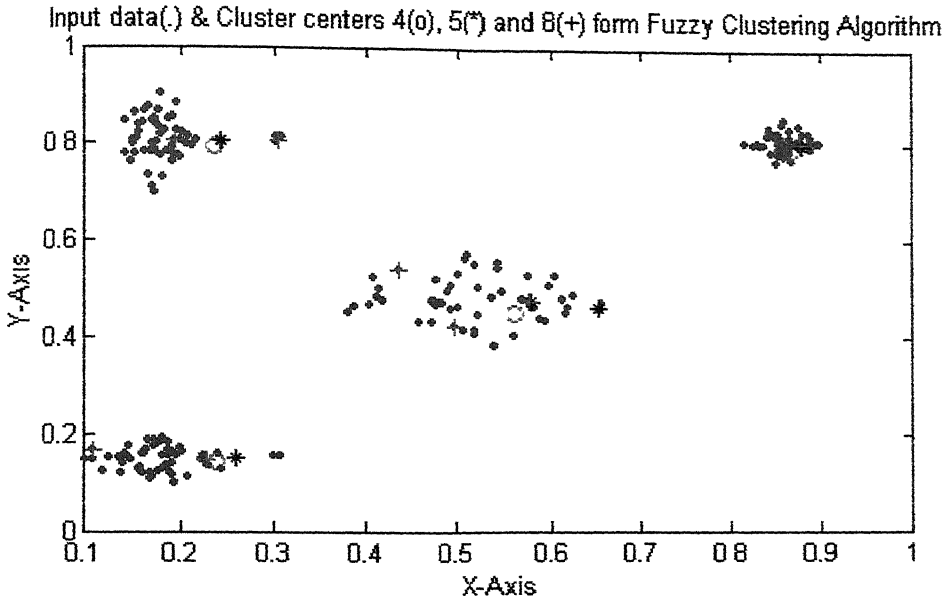


Fig. 5.13 Cluster centers for 4 (o), 5 (*) and 8 (+) clusters with Fuzzy-Clustering Algorithm

Although the cluster centers obtained are not very close to the centers of the ellipses, they are close enough to determine the number of clusters accurately. More rigorous training of cluster centers may place them at the actual cluster centroids.

CHAPTER 6

CONCLUSIONS

6.1 Summary

The investigation begins with development of OR/AND neural network, comprising of compensatory OR and AND operation based neurons. Several OR and AND operations have been implemented along with several aggregation operators. But the general problems associated with ANN, like the requirement of large architectures to solve highly nonlinear functions, slow learning time and getting trapped in local minima are still present. But the severest of all, the random initialization of network weights, was present in case of the OR/AND network also. Because of this the network some times failed to converge.

An attempt has been made to overcome these problems, by first partitioning the input data and then using one smaller network for each of the partitions. For 4-Bit Parity and Sin(x)*Sin(y) problems, the data was partitioned, and one network of OR/AND neurons has been used for each of the partitions. And it was found that not only can the learning time be reduced but the total number of neurons can also be reduced, in some cases. Hence, the need for algorithms that can incorporate the algorithm for partitioning of the input data and the smaller neural network for each partition, in a single architecture, being identified, the main objective of this thesis work has been to develop such algorithms.

Neuro-Fuzzy Systems have been the natural choices, because of their inherent partitioning properties. Two different Neuro-Fuzzy Systems, ANFIS, and CNFS have been implemented. A hybrid learning rule has been employed for ANFIS. A compensatory

reasoning mechanism has been used in case of CNFS Later this has been extended for ANFIS also. When these algorithms were validated on several benchmark problems it was observed that ANFIS performs better for function approximation problems while CNFS performs better for classification problems.

But in both the cases the total number of rules required was not easily controllable. To alleviate this problem, an off-line clustering algorithm has been used to partition the input space and determine the number of rules A new rule-based clustering technique, which automatically finds out the number of clusters, has been proposed. An energy function has been defined to overcome the problem of selection of a suitable vigilance parameter. The energy function is such that when the energy is minimized, the intra-cluster distances are minimized and the inter-cluster distances are maximized.

The Neuro-Fuzzy Systems thus formed are termed Static Generalized Neuro-Fuzzy Systems, because of the offline nature of the clustering algorithm. SG-ANFIS performed better for function problems where as SG-CNFS gave better performance for classification problems, in agreement with the conclusion drawn for the initial versions of ANFIS and CNFS. But these algorithms can not take care of the changes during the training process as the number of rules was determined offline.

An online self organizing algorithm has also been implemented to generate rules dynamically. Different criteria for generation of rules were. This dynamic incorporation of rules, lead to a slower convergence, but gave solutions with less number of rules. Since the number of rules for any problem is automatically decided by the algorithm dynamically, these algorithms are termed Dynamic Generalized Neuro-Fuzzy Systems (DG-NFS).

6.2 Scope for further work

- Validation for the algorithms SG-NFS and DG-NFS was based only on the experimental results and no mathematical explanation as to how these algorithms will work better was presented. An attempt can be made to give mathematical proof.
- In the proposed Rule-Based clustering algorithm, widths of the membership functions have not been modified. A detailed investigation, on updating membership function widths may be of considerable significance, to establish it as a robust clustering technique.
- Many heuristics are used in DG-NFS, to generate rules. These may not be sufficient for different classes of problems. There is scope to standardize the rule generalization criteria.

References

[1] K. Hirota and W. Pedrycz, "OR/AND Nuron in Modeling Fuzzy Set Connectives," *IEEE Trans Fuzzy Syst*, vol. 2, pp 151-161, 1994.

[2] G. J. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic-Theory and Applications*, New Jersey, Prentice-Hall, 1995.

[3] S. Kumar, "Comparison of Performance of Artificial Neural Network on Typical Benchmark Problems using Matlab Tools vis-à-vis Codes written in Java," M.Tech. Thesis, IIT, Kanpur, Jan 2001.

[4] J.-S. R. Jang, "ANFIS: Adaptive-Network-Based Fuzzy Inference System," *IEEE Trans Syst, Man, Cybern.*, vol. 23, pp. 665-685, May/June 1993.

[5] Y. Zhang and A Kandel, "Compensatory Neurofuzzy Systems with Fast Learning Algorithms," *IEEE Trans. Neural Networks,* vol. 9, pp. 83-105, Jan 1998.

[6] W. Shiqian and J. E. Meng, "A Fast Approach for Automatic Generation of Fuzzy Rules by Generalized Dynamic Neural Networks," *IEEE Trans. Fuzzy Syst.*, vol. 9, pp. 578-594, Aug 2001.

[7] J. M. Zurada, *Introduction to Artificial Neural Networks*, Delhi, Jaico Publishing House, 1994.

[8] M. H. Hassoun, *Fundamentals of Artificial Neural Networks*, New Delhi, Prentice Hall of India, 1998.

[9] T. J. Ross, *Fuzzy Logic with Engineering Applications*, Singapore, McGraw-Hill, Inc., 1997.

Appendix A

A. **Derivatives for different T-Norms**

The derivatives for T-Norms are presented here. The derivatives for T-Co Norms can be found similarly. All the formulae presented are for two-inputs (a,b)only. Some of them have an additional parameter p. The derivatives with respect to a and p (where ever apply), are given below. Interchange of variables, a and b give the derivatives for b.

A.1 Product, ab

$$da = b$$

A.2 Derived from averaged sum OR, $\frac{1}{2}\left(1-\left(\frac{1}{2}(1-a)+\frac{1}{2}(1-b)\right)\right)$

$$da = 1$$

A.3 Derived from probabilistic sum OR, $1-(1-a)-(1-b)+(1-a)(1-b)$

$$da = b$$

A.4 Dombi, $\left(1+\left(\left(\frac{1}{a}-1\right)^{p}+\left(\frac{1}{b}-1\right)^{p}\right)^{\frac{1}{p}}\right)^{-1}$

with

$$Y=\left(\left(\frac{1}{a}-1\right)^{p}+\left(\frac{1}{b}-1\right)^{p}\right)$$

$$X=1+Y^{1/p},$$

$$da = \frac{-1}{X^2}\left[Y^{1/p-1}\left(\frac{1}{a}-1\right)^{p-1}\left(\frac{-1}{a^2}\right)\right]$$

$$dp = \frac{-1}{X^2}\left[(X-1)\log Y\left(\frac{-1}{p^2}\right)\left[\left(\frac{1}{a}-1\right)^p \log\left(\frac{1}{a}-1\right)+\left(\frac{1}{b}-1\right)^p \log\left(\frac{1}{b}-1\right)\right]\right]$$

A 5 Frank, $\log_p\left[1+\frac{(p^a-1)(p^b-1)}{p-1}\right]$

with

$$X = 1+\frac{(p^a-1)(p^b-1)}{p-1},$$

$$da = \frac{1}{X}\left[\frac{(p^a-1)}{p-1}p^a\right]$$

$$dp = \frac{\log p\left[\frac{1}{X}\frac{(p-1)\left[(p^a-1)b(p^{b-1})+(p^b-1)a(p^{a-1})\right]-(p^a-1)(p^b-1)}{(p-1)^2}\right]-\frac{1}{p}\log X}{(\log p)^2}$$

A.6 Hamacher, $\frac{ab}{p+(1-p)(a+b-ab)}$

with

$$X = p+(1-p)(a+b-ab),$$

$$da = \frac{Xb-ab(1-p)(1-b)}{X^2}$$

$$dp = \frac{-ab}{X^2}[1-(a+b-ab)]$$

A.7 Schweizer & Sklar2, $1-\left[(1-a)^p+(1-b)^p-(1-a)^p(1-b)^p\right]^{\frac{1}{p}}$

with

$$X = \left[(1-a)^p+(1-b)^p-(1-a)^p(1-b)^p\right],$$

$$da = X^{\frac{1}{p}-1}(1-a)^{p-1}\left[1-(1-b)^p\right]$$

$$dp = -X^{\frac{1}{p}}\log X\left(\frac{-1}{p^2}\right)\left[(1-a)^p\log(1-a)+(1-b)^p\log(1-b)-(1-a)^p(1-b)^p\log((1-a)(1-b))\right]$$

A.8 Schweizer & Sklar3, $\exp\left(-\left(|\ln a|^{p}+|\ln b|^{p}\right)^{\frac{1}{p}}\right)$

with

$$Y=\left(|\ln a|^{p}+|\ln b|^{p}\right)$$

$$X=-Y^{1/p},$$

$$da=-e^{-Y}Y^{1/p-1}|\ln a|^{p-1}\frac{1}{a}$$

$$dp=-e^{-Y}Y^{1/p}\log Y\left(\frac{-1}{p^{2}}\right)\left(|\ln a|^{p}\log|\ln a|+|\ln b|^{p}\log|\ln b|\right)$$

A 9 Schweizer & Sklar4, $\dfrac{ab}{\left[a^{p}+b^{p}-a^{p}b^{p}\right]^{\frac{1}{p}}}$

with

$$X=a^{p}+b^{p}-a^{p}b^{p},$$

$$da=\frac{X^{1/p}b-X^{1/p-1}a^{p}\left(1-b^{p}\right)}{\left(X^{1/p}\right)^{2}}$$

$$dp=\frac{-ab}{\left(X^{1/p}\right)^{2}}X^{1/p-1}\left[a^{p-1}+b^{p-1}-a^{p-1}b^{p-1}\right]$$

Appendix B

Table B.1 Data for XOR Problem

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Appendix C

Results obtained for 4-Bit Parity Problem using OR/AND Network

Table C.1 Comparison of Outputs obtained form Single Network and 4-Networks

4-Bit Parity Problem. Architecture used: (2-1); error Tolerance 1e-12 Architecture for 4-Networks: (2-1)						
X1	X2	X3	X4	Desired Output	Output from Single Network	**Output from 4-Networks**
0.1	0.1	0 1	0 1	0.1	0 1000006686	**0.0999986316**
0.1	0.1	0.1	0.9	0.9	0 9000004458	**0.9000000442**
0.1	0.1	0.9	0.1	0.9	0.9000002503	**0.9000000605**
0.1	0.1	0.9	0.9	0.1	0.0999999591	**0.1000003061**
0.1	0.9	0.1	0.1	0.9	0.9000004394	**0.899999458**
0.1	0.9	0.1	0 9	0.1	0.1000001448	**0.099999985**
0.1	0.9	0.9	0.1	0.1	0.0999999568	**0.099999948**
0.1	0.9	0 9	0.9	0.9	0.8999997123	**0.900001299**
0.9	0.1	0.1	0.1	0.9	0.9000003489	**0.899999415**
0.9	0.1	0.1	0 9	0.1	0.1000000494	**0.099999989**
0.9	0.1	0.9	0.1	0.1	0.0999998615	**0.099999953**
0.9	0.1	0.9	0 9	0.9	0.8999996218	**0.900001279**
0.9	0.9	0.1	0.1	0.1	0.1000000472	**0.0999988343**
0.9	0.9	0.1	0.9	0.9	0.8999998109	**0.9000000347**
0.9	0.9	0.9	0.1	0.9	0.8999996153	**0.8999998756**
0.9	0.9	0.9	0.9	0.1	0.0999993377	**0.1000007514**

Appendix D

Table D 1 Data for 4-Ellipse Problem

Sl No	X1	X2	Sl No	X1	X2	Sl No	X1	X2
1	0 5767542	0 4766458	46	0 8541026	0 8061659	91	0 4766919	0 4688385
2	0 2264824	0 1482809	47	0 1574846	0 8362011	92	0 1610332	0 1170515
3	0 8768346	0 8050107	48	0 4352669	0 5398982	93	0 8514203	0 7633715
4	0 1944454	0 8050107	49	0 1602476	0 1598772	94	0 1712632	0 7087202
5	0 6123051	0 4790465	50	0 8443855	0 8376911	95	0 5017439	0 4632063
6	0 1964128	0 1535968	51	0 1534406	0 8629921	96	0 1712211	0 1119942
7	0 9	0 806897	52	0 4079663	0 5234752	97	0 8536777	0 7794756
8	0 2091266	0 8116984	53	0 1456414	0 1745054	98	0 1740226	0 7821635
9	0 5479823	0 4977416	54	0 8415541	0 8199961	99	0 508369	0 4138334
10	0 1876354	0 1557665	55	0 1705524	0 845284	100	0 1699931	0 1168747
11	0 8793673	0 8111353	56	0 4927611	0 5072054	101	0 8573392	0 7817975
12	0 2021163	0 8223637	57	0 1363454	0 1514977	102	0 1742174	0 6971372
13	0 6261718	0 4867191	58	0 8431292	0 8347661	103	0 5240596	0 4466051
14	0 2407552	0 1603688	59	0 1631279	0 8661299	104	0 1768995	0 1223366
15	0 8686094	0 8060181	60	0 4757761	0 5201483	105	0 8611614	0 7981833
16	0 1960743	0 8241499	61	0 1452773	0 1542131	106	0 1756925	0 8009142
17	0 6049178	0 5280579	62	0 8493222	0 8241254	107	0 5405642	0 3825691
18	0 1995542	0 1700322	63	0 1499563	0 8056698	108	0 1826271	0 1281216
19	0 8942411	0 8069881	64	0 4712779	0 4766458	109	0 8646835	0 7996349
20	0 1930036	0 8531272	65	0 1086982	0 1654054	110	0 1800257	0 7740998
21	0 5376865	0 4846878	66	0 8427938	0 8216316	111	0 5195918	0 4089822
22	0 2244684	0 1563229	67	0 14345	0 8437927	112	0 1894132	0 1183528
23	0 8871906	0 8251157	68	0 4135165	0 4991025	113	0 8682503	0 7672753
24	0 1804069	0 8170737	69	0 1421239	0 1595093	114	0 1867732	0 7828899
25	0 5979123	0 5084898	70	0 8586803	0 8073925	115	0 5205892	0 4147472
26	0 1814509	0 154837	71	0 1566048	0 8193014	116	0 1934108	0 1
27	0 8890405	0 8162497	72	0 4131226	0 4855627	117	0 859135	0 7765373
28	0 2001341	0 8068839	73	0 1264871	0 1525679	118	0 1925014	0 7604438
29	0 54418	0 5567655	74	0 8300743	0 806897	119	0 5604398	0 4027132
30	0 1929215	0 1630398	75	0 155093	0 8087832	120	0 2089837	0 1090042
31	0 8811436	0 8250406	76	0 4187048	0 4766458	121	0 866119	0 7853723
32	0 196325	0 8809136	77	0 1090133	0 1482809	122	0 1975394	0 7726651
33	0 5695951	0 4812666	78	0 8170321	0 8050107	123	0 5877453	0 4386945
34	0 1872396	0 1667641	79	0 1517293	0 8050107	124	0 1867952	0 1293052
35	0.876991	0 8431322	80	0 4046468	0 4680718	125	0 8722945	0 8050107
36	0.1872396	0 8512187	81	0 1	0 1479379	126	0 1813495	0 7301622
37	0 5760615	0 5286407	82	0 8364312	0 8027815	127	0 5947985	0 435436
38	0 203414	0 1618986	83	0.1497956	0.7986659	128	0 1931279	0 1126906
39	0 8741398	0 8074867	84	0 4817838	0 4712017	129	0 8882621	0 8050107
40	0.184317	0 8272942	85	0 1390209	0 1465796	130	0 2009126	0 7684839
41	0 5453317	0 5443094	86	0 8276497	0 8026289	131	0 6171701	0 4509115
42	0 1814931	0 1925225	87	0 1421239	0 7781306	132	0 1855906	0 1354137
43	0 8662702	0 8323364	88	0 3878735	0.4625432	133	0 8785512	0 7961645
44	0 1805032	0 9	89	0 1414642	0 1382076	134	0 2063254	0 7961645
45	0 5197532	0 5519072	90	0 8557016	0 7944337	135	0 6554517	0 4608268

Table D 1 Contd

Sl No	X1	X2	Sl No	X1	X2	Sl No	X1	X2
136	0 1872294	0 1818797	161	0 1702586	0 7823458	186	0 2435373	0 1271889
137	0 8658841	0 8184503	162	0 3803236	0 4515989	187	0 8736533	0 8023742
138	0 1772242	0 8641446	163	0 1190746	0 1225748	188	0 1930036	0 7786456
139	0 5232548	0 5039555	164	0 8540012	0 791169	189	0 6202145	0 4625433
140	0 1802522	0 1837836	165	0 1490205	0 760849	190	0 1930956	0 1402223
141	0 8596861	0 8514373	166	0 4962169	0 4251769	191	0 8894606	0 7949376
142	0 1777042	0 8336859	167	0 1579442	0 1281759	192	0 2139467	0 7949376
143	0 5102463	0 564037	168	0 8552245	0 7897309	193	0 5831503	0 4643968
144	0 1730104	0 1728597	169	0 1536244	0 7768637	194	0 2300438	0 1356916
145	0 8580768	0 8446098	170	0 4920408	0 4616604	195	0 8659207	0 7941227
146	0 173748	0 8432444	171	0 1495338	0 1473443	196	0 1928071	0 7931019
147	0 5013435	0 5304039	172	0 8563449	0 7928351	197	0 6568033	0 4718445
148	0 1632169	0 1576886	173	0 1635462	0 7825326	198	0 2228998	0 1467376
149	0 8516766	0 8171063	174	0 4774708	0 472429	199	0 8872863	0 8034674
150	0 1736223	0 8480172	175	0 1384651	0 1198173	200	0 2165429	0 8034674
151	0 5110117	0 5703339	176	0 8334942	0 8029023	201	0 3055072	0 1537537
152	0 1682931	0 1886188	177	0 1718196	0 7955229	202	0 307643	0 1537537
153	0 8563703	0 8154205	178	0 457442	0 4304378	203	0 2990998	0 1537537
154	0 1682931	0 8726744	179	0 1597735	0 1321081	204	0 3055072	0 15649
155	0 4742108	0 4766458	180	0 8449473	0 7842171	205	0 3055072	0 15649
156	0 1656746	0 1680885	181	0 1683567	0 7322332	206	0 3033714	0 8050107
157	0 8527693	0 8235803	182	0 4714828	0 4320788	207	0 307643	0 8132199
158	0 1602184	0 8396739	183	0 1688575	0 1074278	208	0 3055072	0 8132199
159	0 4883412	0 4974394	184	0 8514053	0 7641576	209	0 3012356	0 8132199
160	0 174651	0 1887129	185	0 1738591	0 7864412	210	0 3012356	0 799538

Table D.2 Clustering process for 4-Ellipse problem with Fuzzy-Classifier

Iteration Number	Minimum Firing Strength	Number of Clusters	Distance		System Energy
			Inter-Cluster	Intra-Cluster	
0	0.0	1	0 001	33.83	1033.83
1	0.1	1	0 001	33.83	1033 83
2	0.2	1	0.001	33.83	1033.83
3	0 3	2	0.375	17 016	19 681
4	0.4	3	0.283	6.786	10.319
5	0 5	4	0.201	1.139	6.12
6	0.6	4	0 201	1.139	6.12
7	0.7	4	0.201	1 139	6.12
8	0 8	4	0.201	1.139	6.12
9	0.9	4	0.201	1.139	6.12
10	1.0	209	6.18e-07	0.0	1.61e06
11	0.9	4	0.201	1.139	6.12
12	0.91	4	0.201	1.139	6.12
13	0.92	4	0.201	1.139	6.12
14	0.93	4	0.201	1.139	6.12
15	0.94	5	0.005	1.321	173.5
16	**0.93**	**4**	**0.201**	**1.139**	**6.12**

Appendix E

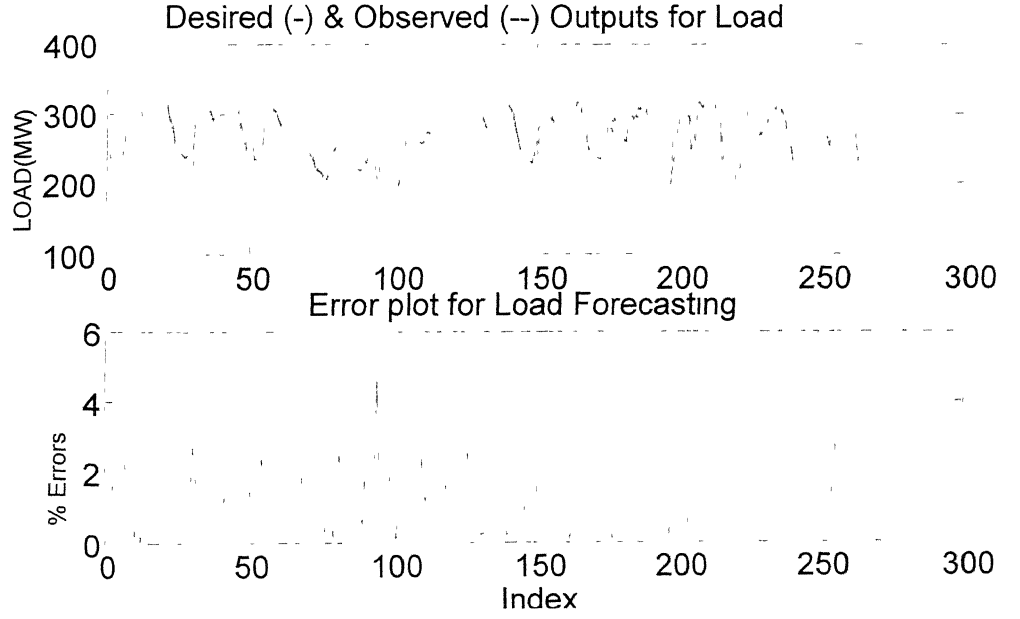
RESULTS FOR LOAD-FORECASTING PROBLEM

Electrical Load-Forecasting problem has also been considered, where the hourly load was predicted. To reflect the various factors affecting the hourly load (like temperature, moisture etc.), the inputs considered were,

1. The previous hour load (L-1)
2. Previous to previous hour load (L-2)
3. Previous day same hour load (L-24)
4. Previous day previous to previous hour load (L-25)
5. Previous week same hour load (L-168).

For training the system, only 176 patterns were presented, and 86 patterns were used for testing. The SG-ANFIS with 29 rules was used to predict the load, which took 143 epochs to train to an accuracy of 0.0019, giving a test error of 0.0034.

Fig E.1 Simulation Results for Load-Forecasting Problem (SG-ANFIS)



The Function Approximation problem

A complicated multivariable function in seven dimensions has been selected for this problem. The function is as described below.

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = \sum_{i=1,3,5} \sin(x_i)\sin(x_{i+1})\exp\left(-(x_i^2 + x_{i+1}^2)\right)$$

$$(-1 \leq x_1, x_2, x_3, x_4, x_5, x_6 \leq 1)$$

The SG-CNFS algorithm was used to solve this problem. Its results were compared with those obtained for a Standard Neural Network. The SG-CNFS used has 40 rules, and took 348 epochs to get trained to an accuracy of 1.7e-04, whereas the standard ANN took 4424 epochs, with architecture of (9-51-49-45-1), to an accuracy of 1.4e-03.

Fig. E.2 Simulation results for six input nonlinear function with standard ANN

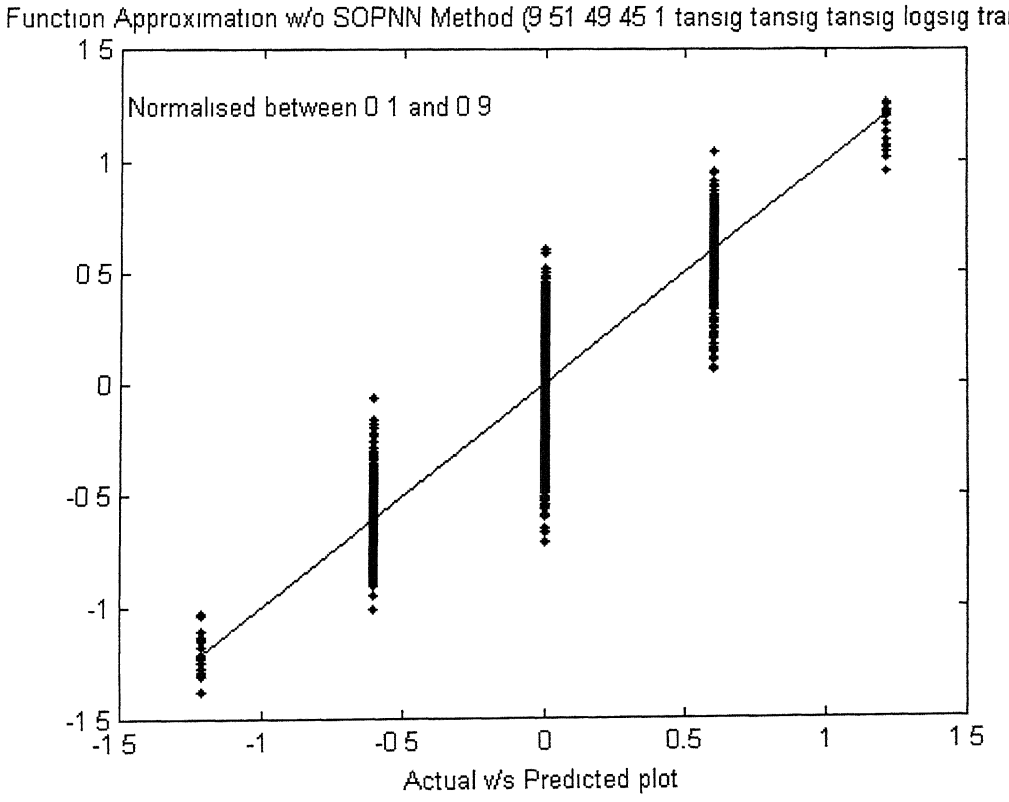
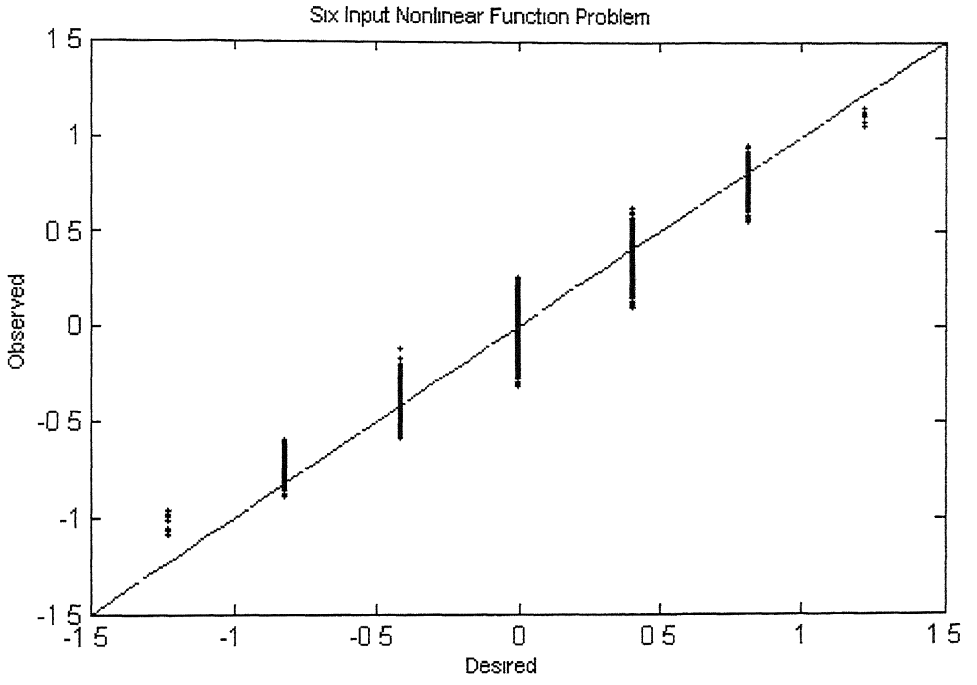


Fig. E.3 Simulation results for six input nonlinear function with SG-CNFS



Appendix F

Comparison of the various algorithm complexities

The complexity of an algorithm can be either the weight complexity or the time complexity indicated by the number of operations per pattern presented while testing. The weight complexity varies from problem to problem, where as the time complexity for a pattern per neuron or per rule remains constant. The time complexities for different algorithms are presented below.

Table F.1 Comparison of time complexities for various algorithms

Algorithm	Number of operations per pattern per neuron (or per rule)		
	Multiplications	Additions	Nonlinear Calculations
OR/AND	3*Inputs	1*Inputs	Sigmoid
ANFIS	1*Inputs+1	1*Inputs+1	Inputs*Gaussian
CNFS	1	1	(Inputs+Outputs)*Gaussian

The complexities for the other Neuro-Fuzzy algorithms will be same as those of ANFIS and CNFS, since once trained, their architectures are equivalent. As can be seen form the above table, the number of multiplications and additions increase linearly with the number of inputs in ANFIS and OR/AND Network, while remaining constant in case of CNFS. Nevertheless, the number of calculations of nonlinear membership functions is high in CNFS.

Since the weight complexities are dependent on the problem being considered, these are presented for the 4-Bit Parity and Sin(x)Sin(y) problems.

Table F.2 Comparison of weight complexities of various algorithms

Problem	Algorithm	Number of weights	Performance Index
4-Bit Parity	OR/AND (4-1)	12	1e-04
	ANFIS (4 rules)	28	1e-12
	CNFS (16 rules)	40	1e-12
Sin(x)Sin(y)	OR/AND (2-12-8-1)	229	1e-04
	ANFIS (16 rules)	56	2.3e-05
	CNFS (16 rules)	96	7.4e-05

From the above table it is clear that the number of weights reduce for ANFIS and CNFS as the complexity of the [problem considered increases. It can be concluded that although the number of weights for the 4-Bit Parity problem in case of the Neuro-Fuzzy Systems increased, so was the performance index. The improvement in both the number of weights and the performance index is evident in case of the Sin(x)Sin(y) problem.